

**Space Project Mission Operations Control  
Architecture (SuperMOCA)**

**SuperMOCA SYSTEM CONCEPT**

**Annex 2**

**Space Messaging Service (SMS)  
Service Specification**

April 1996

## SuperMOCA Operations Concept



# ***ORIENTATION***

The goal of the Space Project Mission Operations Control Architecture ("SuperMOCA") is to create a set of implementation-independent open specifications for the standardized monitor and control of space mission systems. Monitoring is the observation of the performance of the activities of these systems. Controlling is the direction of the activities performed by these systems. Overall, monitor and control is the function that orchestrates the activities of the components of each of the systems so as to make the mission work. Space mission systems include:

spacecraft and launch vehicles that are in flight, and;  
their supporting ground infrastructure, including launch pad facilities and ground terminals used for tracking and data acquisition.

The SuperMOCA system concept documents consist of the following:

SuperMOCA System Concept, Volume 1: Rationale and Overview  
SuperMOCA System Concept, Volume 2: Architecture  
SuperMOCA System Concept, Volume 3: Operations Concepts  
SuperMOCA System Concept, Annex 1: Control Interface Specification  
SuperMOCA System Concept, Annex 2: Space Messaging Service (SMS) Service Specification  
SuperMOCA System Concept, Annex 3: Communications Architecture  
SuperMOCA System Concept, Ancillary Document 1: Ground Terminal Reference Model  
SuperMOCA System Concept, Ancillary Document 2: Operations Center to Ground Terminal Scenarios  
SuperMOCA System Concept, Ancillary Document 3: Operations Center to Ground Terminal – Comparison of Open Protocols

These documents are maintained by the custodian named below. Comments and questions to the custodian are welcomed.

Michael K. Jones  
MS 301-235  
Jet Propulsion Laboratory  
4800 Oak Grove Drive  
Pasadena, CA 91109  
Voice: 818-354-3918  
FAX: 818-354-9068  
E-mail: michael.k.jones@jpl.nasa.gov

## Contents

SECTION	PAGE
<b><u>1.</u> OBJECT OF THE STANDARD .....</b>	<b>1-1</b>
1.1. PURPOSE.....	1-1
1.2. SCOPE.....	1-1
1.3. APPLICABILITY .....	1-2
1.4. OVERVIEW.....	1-2
<b><u>2.</u> NORMATIVE REFERENCES .....</b>	<b>2-1</b>
<b><u>3.</u> DEFINITIONS.....</b>	<b>3-1</b>
3.1. REFERENCE MODEL DEFINITIONS .....	3-1
3.2. SERVICE CONVENTIONS DEFINITIONS .....	3-1
3.3. ABSTRACT SYNTAX NOTATION DEFINITIONS .....	3-2
3.4. OTHER DEFINITIONS .....	3-2
<b><u>4.</u> ABBREVIATIONS.....</b>	<b>4-1</b>
<b><u>5.</u> CONVENTIONS.....</b>	<b>5-1</b>
5.1. BASE OF NUMERIC VALUES.....	5-1
5.2. SERVICE PARAMETER DESCRIPTION .....	5-1
5.2.1. Companion Standard Service Parameters .....	5-1
5.2.2. Service Table Structure.....	5-1
5.3. INVOCATION IDENTIFIER ON SERVICE PRIMITIVES .....	5-3
5.4. LIST OF MODIFIER ON SERVICE PRIMITIVES .....	5-3
5.5. ADDRESSING IN SMS .....	5-3
5.6. SERVICE CONVENTIONS .....	5-4
5.7. CALLING AND CALLED SMS-USER .....	5-4
5.8. SENDING AND RECEIVING SMS-USER AND SMPM .....	5-4
5.9. REQUESTING AND RESPONDING SMS-USER.....	5-5
5.10. CLIENT AND SERVER OF A SERVICE.....	5-5
5.11. OBJECT MODELING.....	5-6
5.12. REFERENCE TO OBJECTS.....	5-7
5.13. PARAMETER TYPES.....	5-7
<b><u>6.</u> THE SMS ENVIRONMENT.....</b>	<b>6-1</b>
6.1. INFORMATION PROCESSING TASKS AND REAL SYSTEMS .....	6-1
6.2. APPLICATION PROCESSES .....	6-2
6.3. INTERACTION OF APPLICATION PROCESSES .....	6-2
6.4. INTERACTION OF APPLICATION PROCESSES IN SPACE .....	6-3
6.5. STRUCTURE OF APPLICATION ENTITIES .....	6-3
6.6. ADDRESSING OF APPLICATION ENTITIES.....	6-4
6.7. APPLICATION CONTEXT .....	6-4
6.8. PRESENTATION CONTEXT, ABSTRACT SYNTAXES, AND TRANSFER SYNTAXES .....	6-5
6.9. LOWER LAYER PROTOCOL STACK CONSIDERATIONS.....	6-5
6.9.1. Full Stack.....	6-6
6.9.2. Enhanced Performance Architecture .....	6-8

## SuperMOCA Space Messaging Service

6.9.3.	Inter-task Communications.....	6-9
<b>7.</b>	<b>THE VIRTUAL SPACECRAFT DEVICE MODEL .....</b>	<b>7-1</b>
7.1.	INTRODUCTION .....	7-1
7.1.1.	Relationship of the VSD to the OSI Model .....	7-1
7.1.2.	Relationship of the VSD to the Real Spacecraft Device.....	7-2
7.2.	THE STRUCTURE OF A VSD.....	7-3
7.2.1.	The Executive Function .....	7-3
7.2.2.	Vendor Name.....	7-3
7.2.3.	Model Name.....	7-3
7.2.4.	Revision .....	7-4
7.2.5.	List Of Abstract Syntaxes Supported .....	7-4
7.2.6.	Logical Status.....	7-4
7.2.7.	List of Capabilities .....	7-5
7.2.8.	Physical Status .....	7-6
7.2.9.	List Of Program Invocations .....	7-6
7.2.10.	List Of Domains .....	7-7
7.2.11.	Transaction Object .....	7-7
7.2.12.	Upload State Machines .....	7-9
7.2.13.	Other VMD-specific Named Objects.....	7-10
7.2.14.	Additional Detail .....	7-10
7.3.	SPECIFICATION OF NAMED OBJECTS.....	7-10
7.3.1.	Scope of Names.....	7-10
7.3.2.	Classes of Objects.....	7-10
7.3.3.	Object Lifetime.....	7-11
7.3.4.	Object Visibility.....	7-12
7.3.5.	Creation of SMS Objects .....	7-12
7.3.6.	Deletion of SMS Objects.....	7-12
7.3.7.	Alteration of SMS Objects .....	7-12
7.4.	OBJECT NAME STRUCTURE.....	7-13
7.4.1.	Name Scope .....	7-13
7.4.2.	AA-specific .....	7-13
7.4.3.	Domain-specific.....	7-13
7.4.4.	VSD-specific .....	7-14
7.5.	SERVICES ON THE VSD .....	7-14
<b>8.</b>	<b>GENERAL MANAGEMENT SERVICES .....</b>	<b>8-1</b>
8.1.1.	Introduction .....	8-1
8.1.2.	Environment Management State Diagram.....	8-1
8.2.	INITIATE SERVICE .....	8-4
8.2.1.	Structure .....	8-5
8.2.2.	Service Procedure .....	8-7
8.2.3.	Init Request Detail Parameter.....	8-8
8.2.4.	Init Response Detail Parameter .....	8-9
8.3.	CONCLUDE SERVICE .....	8-11
8.3.1.	Structure .....	8-11
8.3.2.	Service Procedure .....	8-11
8.4.	ABORT SERVICE.....	8-13
8.4.1.	Structure .....	8-13

## SuperMOCA Space Messaging Service

8.4.2.	Service Procedure .....	8-14
8.5.	CANCEL SERVICE.....	8-14
8.5.1.	Structure .....	8-15
8.5.2.	Service Procedure .....	8-16
8.6.	REJECT SERVICE.....	8-16
8.6.1.	Structure .....	8-16
8.6.2.	Service Procedure .....	8-22
8.7.	REBOOT SERVICE.....	8-22
8.7.1.	Structure .....	8-22
8.7.2.	Service Procedure .....	8-23
<b>9.</b>	<b>VIRTUAL DEVICE SUPPORT SERVICES.....</b>	<b>9-1</b>
9.1.	INTRODUCTION .....	9-1
9.2.	STATUS SERVICE.....	9-1
9.3.	UNSOLICITED STATUS SERVICE.....	9-1
9.4.	GETNAMELIST SERVICE.....	9-1
9.5.	IDENTIFY SERVICE .....	9-1
9.6.	RENAME SERVICE.....	9-1
9.7.	GETCAPABILITYLIST SERVICE.....	9-1
<b>10.</b>	<b>DOMAIN MANAGEMENT SERVICES.....</b>	<b>10-1</b>
10.1.	THE DOMAIN OBJECT .....	10-1
10.2.	INITIATEDOWNSEQUENCE.....	10-1
10.3.	DOWNLOADSEGMENT.....	10-1
10.4.	TERMINATEDOWNSEQUENCE .....	10-1
10.5.	INITIATEUPLOADSEQUENCE.....	10-1
10.6.	UPLOADSEGMENT.....	10-1
10.7.	TERMINATEUPLOADSEQUENCE.....	10-1
10.8.	REQUESTDOMAINDOWNLOAD .....	10-1
10.9.	REQUESTDOMAINUPLOAD .....	10-1
10.10.	LOADDOMAINCONTENT .....	10-1
10.11.	STOREDOMAINCONTENT .....	10-1
10.12.	DELETEDOMAIN .....	10-1
10.13.	GETDOMAINATTRIBUTES.....	10-1
<b>11.</b>	<b>PROGRAM INVOCATION SERVICES .....</b>	<b>11-1</b>
11.1.	THE PROGRAM INVOCATION OBJECT .....	11-1
11.2.	CREATEPROGRAMINVOCATION.....	11-1
11.3.	DELETEPROGRAMINVOCATION .....	11-1
11.4.	START .....	11-1
11.5.	STOP.....	11-1
11.6.	RESUME .....	11-1
11.7.	RESET.....	11-1
11.8.	KILL .....	11-1
11.9.	GETPROGRAMINVOCATIONATTRIBUTES .....	11-1
<b>12.</b>	<b>VARIABLE ACCESS SERVICES .....</b>	<b>12-1</b>
12.1.	THE SMS VARIABLE ACCESS OBJECT.....	12-1
12.2.	SPECIFICATION OF TYPE.....	12-1

## SuperMOCA Space Messaging Service

12.3.	SPECIFICATION OF ALTERNATE ACCESS .....	12-1
12.4.	SPECIFICATION OF DATA VALUES.....	12-1
12.5.	SPECIFICATION OF ACCESS TO VARIABLE .....	12-1
12.6.	READ .....	12-1
12.7.	WRITE .....	12-1
12.8.	INFORMATIONREPORT .....	12-1
12.9.	GETVARIABLEACCESSATTRIBUTES.....	12-1
12.10.	DEFINENAMEDVARIABLE .....	12-1
12.11.	DEFINESCATTEREDACCESS.....	12-1
12.12.	GETSCATTEREDACCESSATTRIBUTES .....	12-1
12.13.	DELETEVARIABLEACCESS .....	12-1
12.14.	DEFINENAMEDVARIABLELIST.....	12-1
12.15.	GETNAMEDVARIABLELISTATTRIBUTES .....	12-1
12.16.	DELETENAMEDVARIABLELIST.....	12-1
12.17.	DEFINENAMEDTYPE .....	12-1
12.18.	GETNAMEDTYPEATTRIBUTES .....	12-1
12.19.	DELETENAMEDTYPE .....	12-1
12.20.	CONFORMANCE .....	12-1
12.21.	GUIDANCE TO IMPLEMENTORS .....	12-1
<b><u>13.</u></b>	<b>SEMAPHORE MANAGEMENT SERVICES .....</b>	<b>13-1</b>
13.1.	THE SEMAPHORE MANAGEMENT OBJECT .....	13-1
13.2.	TAKECONTROL.....	13-1
13.3.	RELINQUISHCONTROL.....	13-1
13.4.	DEFINESEMAPHORE .....	13-1
13.5.	DELETESEMAPHORE .....	13-1
13.6.	REPORTSEMAPHORESTATUS.....	13-1
13.7.	REPORTPOOLSEMAPHORESTATUS .....	13-1
13.8.	REPORTSEMAPHOREENTRYSTATUS.....	13-1
13.9.	ATTACHTOSEMAPHOREMODIFIER .....	13-1
13.10.	CONFORMANCE .....	13-1
<b><u>14.</u></b>	<b>OPERATOR COMMUNICATION SERVICES.....</b>	<b>14-1</b>
14.1.	THE OPERATOR COMMUNICATIONS MODEL.....	14-1
14.2.	INPUT .....	14-1
14.3.	OUTPUT .....	14-1
<b><u>15.</u></b>	<b>EVENT MANAGEMENT SERVICES.....</b>	<b>15-1</b>
15.1.	THE EVENT MANAGEMENT MODEL .....	15-2
15.2.	DEFINEEVENTCONDITION .....	15-2
15.3.	DELETEEVENTCONDITION .....	15-2
15.4.	GETEVENTCONDITIONATTRIBUTES .....	15-2
15.5.	REPORTEVENTCONDITIONSTATUS.....	15-2
15.6.	ALTEREVENTCONDITIONMONITORING.....	15-2
15.7.	TRIGGEREVENT .....	15-2
15.8.	DEFINEEVENTACTION .....	15-2
15.9.	DELETEEVENTACTION.....	15-2
15.10.	GETEVENTACTIONATTRIBUTES .....	15-2
15.11.	REPORTEVENTACTIONSTATUS .....	15-2

## SuperMOCA Space Messaging Service

15.12.	DEFINEEVENTENROLLMENT .....	15-2
15.13.	DELETEEVENTENROLLMENT .....	15-2
15.14.	GETEVENTENROLLMENTATTRIBUTES.....	15-2
15.15.	REPORTEVENTENROLLMENTSTATUS.....	15-2
15.16.	ALTEREVENTENROLLMENT .....	15-2
15.17.	EVENTNOTIFICATION .....	15-2
15.18.	ACKNOWLEDGEEVENTENROLLMENT .....	15-2
15.19.	GETALARMSUMMARY .....	15-2
15.20.	GETALARMENROLLMENTSUMMARY .....	15-2
15.21.	ATTACHTOEVENTCONDITIONMODIFIER .....	15-2
15.22.	EVENT MANAGEMENT STATE DIAGRAMS .....	15-2
<b>16.</b>	<b>JOURNAL MANAGEMENT SERVICES .....</b>	<b>16-1</b>
16.1.	THE JOURNAL MANAGEMENT MODEL.....	16-1
16.2.	READJOURNAL .....	16-1
16.3.	WRITEJOURNAL.....	16-1
16.4.	INITIALIZEJOURNAL .....	16-1
16.5.	REPORTJOURNALSTATUS.....	16-1
16.6.	CREATEJOURNAL.....	16-1
16.7.	DELETEJOURNAL .....	16-1
16.8.	CONFORMANCE REQUIREMENTS UNIQUE TO JOURNAL.....	16-1
<b>17.</b>	<b>FILE MANAGEMENT SERVICES.....</b>	<b>17-1</b>
17.1.	THE SMS FILE MODEL .....	17-1
17.2.	FILEOPEN .....	17-1
17.3.	FILEREADRECORD.....	17-1
17.4.	FILEWRITERECORD .....	17-1
17.5.	FILERECORDUPDATE .....	17-1
17.6.	FILECLOSE.....	17-1
17.7.	FILECOPY .....	17-1
17.8.	FILECOPYRESUME.....	17-1
17.9.	FILERENAME .....	17-1
17.10.	FILEDELETE.....	17-1
17.11.	FILEDIRECTORY.....	17-1
17.12.	FILEATTRIBUTES .....	17-1
17.13.	ADDITIONAL SPECIFICATION FOR CONCLUDE AND ABORT SERVICES .....	17-1
<b>18.</b>	<b>ERRORS .....</b>	<b>18-1</b>
<b>19.</b>	<b>SPECIAL ATTACHMENT 1.....</b>	<b>19-1</b>

## Figures

FIGURE	PAGE
FIGURE 1-1 - MESSAGING ENVIRONMENT .....	1-1

## SuperMOCA Space Messaging Service

FIGURE 5-1 - CLIENT-SERVER RELATIONSHIPS.....	5-5
FIGURE 6-1 - IMPLEMENTATION ENVIRONMENTS IN WHICH MMS HAS BEEN SUCCESSFULLY DEPLOYED.....	6-6
FIGURE 6-2 - THE PEER-TO-PEER RELATION FOR GROUND-TO-SPACECRAFT, SPACECRAFT-TO- GROUND AND SPACECRAFT-TO-SPACECRAFT IS SHOWN ABOVE FOR SMS OPERATING OVER THE SCPS STACK. ....	6-7
FIGURE 6-3 - COMMUNICATIONS PATHS AND LAYERS.....	6-7
FIGURE 6-4 - THE PEER-TO-PEER RELATION FOR ONBOARD SYSTEMS USING SMS OVER THE EPA STACK.....	6-8
FIGURE 6-5 - THE PEER-TO-PEER RELATION FOR A GROUND SYSTEM COMMUNICATING WITH AN ONBOARD SYSTEMS USING SMS OVER THE EPA STACK .....	6-9
FIGURE 6-6 - THE PEER-TO-PEER RELATION FOR INTER-TASK COMMUNICATIONS .....	6-9
FIGURE 7-1 - THE SMS SERVER APPLICATION PROCESS .....	7-2
FIGURE 8-1 - THE STATE DIAGRAM FOR ENTERING AND LEAVING THE SMS ENVIRONMENT.....	8-3

## Tables

<b>TABLE</b>	<b>PAGE</b>
TABLE 7-1 - SMS OBJECTS .....	7-11
TABLE 7-2 - OBJECT NAME .....	7-13
TABLE 8-1 - STRUCTURE OF COMPONENT SERVICE PRIMITIVES.....	8-5
TABLE 8-2 - STRUCTURE OF THE INIT REQUEST DETAIL PARAMETER.....	8-8
TABLE 8-3 - STRUCTURE OF THE INIT RESPONSE DETAIL PARAMETER.....	8-9
TABLE 8-4 - CONCLUDE SERVICE .....	8-11
TABLE 8-5 - ABORT SERVICE.....	8-13
TABLE 8-6 - CANCEL SERVICE.....	8-15
TABLE 8-7 - REJECT SERVICE .....	8-16
TABLE 8-8 - REBOOT SERVICE.....	8-22

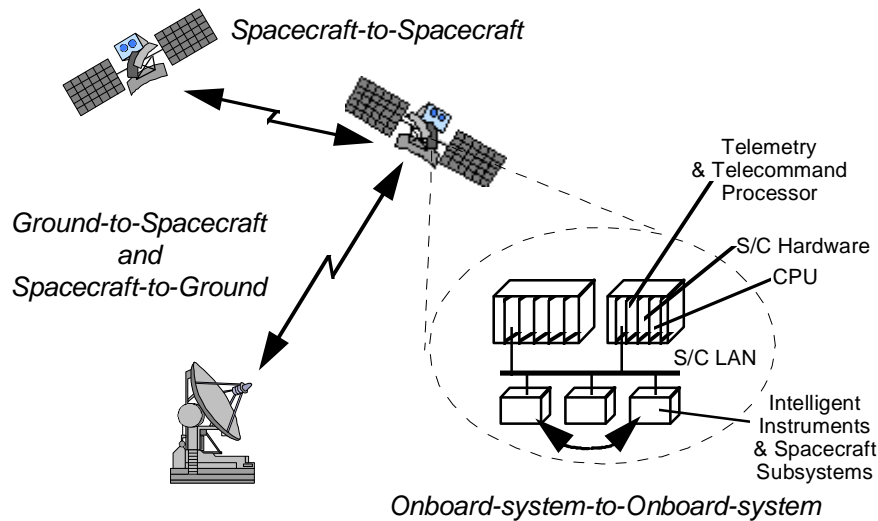
## 1.     **OBJECT OF THE STANDARD**

### **1.1.    PURPOSE**

The purpose of this standard is to define the services and protocols of the Space Messaging Service (SMS). This specification provides for a high level data system interface for the operations of spacecraft, spacecraft devices and instrumentation and to support inter-operation of systems (and subsystems) independent of implementation.

### **1.2.    SCOPE**

The Spacecraft Messaging Service is an application layer standard designed to support ground messaging communications to and from spacecraft, spacecraft messaging communications to and from other spacecraft, and messaging communications to and from devices operating onboard spacecraft (Figure 1-1). The environment referred to in this standard is space, making no distinction with regard to: the complex or simplicity of the spacecraft, manned or unmanned spacecraft, trajectory to near earth orbit, high earth orbit or deep space.



**Figure 1-1 - Messaging Environment**

The standard defines the Space Messaging Service within the Open Systems application layer in terms of:

- a) an abstract model defining the interactions between users of the service;
- b) the externally visible functionality of the implementations conforming to this standard, in the form of procedural requirements associated with the execution of service requests;
- c) the primitive actions and events of the service;
- d) the parameter data associated with each primitive action and event;

- e) the relationship between, and the valid sequences of, these actions and events.

This part of this standard does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces within a computer system, device or instrument. This standard does specify the externally visible functionality of implementations, and specifies conformance requirements for such functionality.

### **1.3. APPLICABILITY**

This standard is designed to be applicable to any kind of space mission regardless of scope and complexity. It is intended to become a uniform standard for all agencies identified by the SuperMOCA Terms of Reference.

### **1.4. OVERVIEW**

[Editor's note: The goal of the effort is to develop a standard messaging service for application to spacecraft devices and instrumentation. It is not intended as a vehicle to re-invent a solution to problem that is in many ways similar to other problems that exist in other industries. Therefore, the approach has been to adopt an existing standard that appears to have most of the characteristics required for the environment and augment the existing standard as needed to address those aspects of the space industry that are not supported in the current standard. After an extensive series of meeting, Manufacturing Message Specification (ISO 9506) was selected as the starting point for the development of the Space Messaging Service.]

The standard provides a wide variety of services useful for the operation of spacecraft, spacecraft devices and instrumentation. It is designed to be used both by itself and in conjunction with companion standards, which describe the application of subsets of these services to particular device or instrument types.

The services provided by the Space Messaging Service range from simple to highly complex. It is not expected that all of these services will be supported by all devices and instruments. The subset to be supported is limited in some cases by companion standards, and in all cases may be limited by the implementor. Characteristics important in the selection of a subset of services to be supported include:

- a) applicability to the service to the device or instrument;
- b) the complexity of the services and requirements;
- c) the complexity of provisions of a particular class of service via the network versus the complexity of the device or instrument.

Some SMS services are quite complex and should be considered as advanced functions. Devices or instruments used in very simple applications often will not require such advanced functions, and hence will not support such SMS services.

This document is part of a set of standards produced to facilitate the inter-operation of space information systems. It is positioned within the application layer of the

## SuperMOCA Space Messaging Service

Open Systems Interconnection Environment as an Application Service Element (ASE) with respect to other related standards by the Basic Reference Model for Open Systems Interconnection (ISO 7498).

The aim of Open Systems Interconnection is to allow, with a minimum of technical agreement outside the interconnection standards, the inter-operation of information processing systems:

- a) from different manufacturers;
- b) under different management;
- c) of different levels of complexity;
- d) of different ages.

## 2.     **NORMATIVE REFERENCES**

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

- |                  |                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| ISO 7498-1:1984, | Information Processing Systems - Open Systems Interconnection - Basic Reference Model, 1984.                                                    |
| ISO 7498-3:1984, | Information Processing Systems - Open Systems Interconnection - Naming and Addressing, 1984.                                                    |
| ISO 8326:1987,   | Information Processing Systems - Open Systems Interconnection - Basic Connection Oriented Session Service Definition.                           |
| ISO 8509,        | Information Processing Systems - Open Systems Interconnection - Service Conventions.                                                            |
| ISO 8571,        | Information Processing Systems - Open Systems Interconnection - File Transfer, Access and Management.                                           |
| ISO 8649,        | Information Processing Systems - Open Systems Interconnection - Association Control Service Element - Service Definition.                       |
| ISO 8650,        | Information Processing Systems - Open Systems Interconnection -. Association Control Service Element - Protocol Specification.                  |
| ISO 8822,        | Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Definition.                            |
| ISO 8824,        | Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).                          |
| ISO 8825,        | Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). |
| ISO 9040,        | Information Processing Systems - Open Systems Interconnection - Virtual Terminal Service Definition.                                            |
| ISO 9041,        | Information Processing Systems - Open Systems Interconnection - Virtual Terminal Protocol Specification.                                        |
| ISO 9506-1       | Manufacturing Message Specification - Service Definition.                                                                                       |
| ISO 9506-2       | Manufacturing Message Specification -Protocol Specification.                                                                                    |
| ISO 9545,        | Information Processing Systems - Open Systems Interconnection – Application Layer Structure                                                     |
| ISO 9594         | Information Processing Systems - Open Systems Interconnection - The OSI Directory Specification.                                                |

## SuperMOCA Space Messaging Service

- IEEE 754:1985, IEEE Standard for Binary Floating-Point Arithmetic.
- CCSDS 710.0-G-0 Consultative Committee for Space Data Systems - Space Communications Protocol Standards - Rationale, Requirements and Application Notes - Draft Green Book, January, 1996.
- CCSDS 717.0-W-1 Consultative Committee for Space Data Systems - Space Communications Protocol Standards - File Protocol - White Book, January, 1996.
- CCSDS 714.0-W-1 Consultative Committee for Space Data Systems - Space Communications Protocol Standards - Transport Protocol - White Book, January, 1996.
- CCSDS 713.0-W-1 Consultative Committee for Space Data Systems - Space Communications Protocol Standards - Network Protocol - White Book, January, 1996.

### 3.     **DEFINITIONS**

The definitions contained in this section make use of abbreviations defined in section 4.

For the purposes of this Standard, the definitions in the following subsections apply.

#### **3.1.   REFERENCE MODEL DEFINITIONS**

This Standard is based on the concepts developed in the Basic Reference Model for Open Systems Interconnection (ISO 7498), and makes use of the following terms defined in that Standard:

- a) application-entity;
- b) application-process;
- c) application service element;
- d) open system;
- e) (N)-protocol;
- f) (N)-protocol-data-unit;
- g) (N)-service-access-point;
- h) (N)-layer;
- I) system;
- j) (N)-user-data.

#### **3.2.   SERVICE CONVENTIONS DEFINITIONS**

This Standard makes use of the following terms defined in the OSI Service Conventions (ISO TR 8509) as they apply to the Space Messaging Service:

- a) confirm;
- b) indication;
- c) primitive;
- d) request;
- e) response;
- f) service primitive;
- g) service provider;
- h) service user.

### **3.3. ABSTRACT SYNTAX NOTATION DEFINITIONS**

This Standard makes use of the following terms defined in the Abstract Syntax Notation One (ASN.1 Specification (ISO 8824):

- a) value;
- b) type;
- c) simple type;
- d) structure type;
- e) component type;
- f) tag;
- g) tagging;
- h) type (or value) reference name;
- i) character string type;
- j) boolean type;
- k) true;
- l) false;
- m) integer type;
- n) bitstring type;
- o) octetstring type;
- p) null type;
- q) sequence type;
- r) sequence-of type;
- s) tagged type;
- t) choice type;
- u) selection type;
- v) any type;
- w) object identifier type;
- x) module;
- y) production;
- z) ASN.1 encoding rules;
- aa) ASN.1 character set.

### **3.4. OTHER DEFINITIONS**

For the purpose of this Standard, the following definitions also apply:

**AA-specific** (Application Association specific): An adjective used to describe an object whose name has a scope that is a single application association (i.e. the name may be referenced only on the application association with respect to which the object was defined).

**attribute**: A data element, having a defined meaning, together with a statement of the set of possible values it may take.

**conformance building block (CBB)**: An atomic unit used to describe SMS conformance requirements.

## SuperMOCA Space Messaging Service

Called SMS-user: The SMS-user that issues the Initiate.response service primitive.

Calling SMS-user: The SMS-user that issues the Initiate.request service primitive.

Client: The peer communicating entity which makes use of the VSD for some particular purpose via a service request instance.

data: Any representation to which meaning is or might be assigned (e.g. characters).

domain: An abstract object that represents a subset of the capabilities of a VSD which is used for a specific purpose.

Domain-specific: An adjective used to describe an object whose name has a scope that is a single domain (i.e. the name can be referenced over all application associations established with the VSD that may reference this domain).

download: The process of transferring the content of a domain, including any subordinate objects, via load data to an SMS-user.

event management: The management of event conditions, event actions, and event enrollments.

file: An unambiguously named collection of information having a common set of attributes.

file operation: The transfer of files between open systems, the inspection, modification or replacement of part of a file's content, or the management of a file and its attributes.

filestore: An organized collection of files, including their attributes and names, residing at a particular open system.

information: The combination of data and the meaning that it conveys.

journal: A set of recorded, time-tagged event transitions, variable data, and/or comments, which may be logically ordered during retrieval.

local matter: A decision made by a system concerning its behavior in the Space Messaging Service that is not subject to the requirements of this Standard.

Space Messaging Protocol Machine (SMPM): An abstract machine that carries out the procedures specified in this part of this Standard.

SMS-environment: A specification of the service elements of SMS and semantics of communication to be used during the lifetime of an application association.

SMS-provider: That part of the application entity that conceptually provides the SMS service through the exchange of SMS PDUs.

SMS-user: That portion of the application process which conceptually invokes the Space Messaging Service.

monitored event: A detected change in the state of an event conditions.

network-triggered event: An event which occurs due to an explicit solicitation by a client.

## SuperMOCA Space Messaging Service

operation station: An abstract object representing equipment associated with a VSD that provides for input/output interaction with an operator.

[Note: MMS defined services for factory automation making provisions for supervisory personnel to intervene in the process and to communicate via the factory communications systems. While unmanned spacecraft will most likely not use these services, SMS retains these services for their potential application in manned spacecraft.]

pre-defined object: An object, whose name is of VSD-specific, Domain-specific or Application Association-specific scope, that is instantiated through the use of some mechanism other than an SMS service.

program invocation: An abstract object representing a dynamic element which most closely corresponds to an execution thread in a multi-tasking environment, which is composed of a set of domains.

Receiving SMPM: The SMPM that receives an SMS PDU.

Receiving SMS-user: The SMS-user that receives an indication or confirmation service primitive.

remote device control and monitoring: The manipulation or inspection of the state of a device attached to the responder of a service request.

semaphore: A conceptual lock associated with a logical or physical resource that permits access to that resource only by an owner of the lock.

semaphore management: The control of semaphores.

Server: The peer communicating entity which behaves as a VSD for a particular service request instance.

Sending SMPM: The SMPM that sends an SMS PDU.

Sending SMS-user: The SMS-user issues a request or response service primitive.

standardized object: An object instantiation, whose name is of VSD-specific or Domain-specific scope, whose definition is provided in this Standard or an SMS Companion Standard.

type: An abstract description of a set of values which may be conveyed by the value of a variable.

upload: The process of transferring the content of a domain, including any subordinate objects, via load data from a remote user, in such a manner as to allow subsequent download.

variable: One or more data elements that are referred to together by a single name or description.

variable access: The inspection or modification of variables or components of variables defined at a VSD.

## SuperMOCA Space Messaging Service

VSD-specific: An adjective used to describe an object whose name has a scope that is a single VSD (i.e. the name may be referenced by all application associations established with the VSD).

unconfirmed service: A SMS service request that does not required a confirmation response for completion of the service procedure.

#### 4.     **ABBREVIATIONS**

AA	: Application Association
ACSE	: Association Control Service Element
AE	: application entity
AP	: application process
ASE	: application service element
ASN.1	: Abstract Syntax Notation One
CBB	: conformance building blocks
MMS	: Manufacturing Message Specification
OSI	: Open Systems Interconnection
PDU	: Protocol Data Unit
PSAP	: presentation service access point
SMS	: Space Messaging Service

## 5.      **CONVENTIONS**

The following conventions apply throughout this Standard.

### **5.1.    BASE OF NUMERIC VALUES**

This Standard uses a decimal representation for all numeric values unless otherwise noted.

### **5.2.    SERVICE PARAMETER DESCRIPTION**

This part of the Standard uses a tabular format to describe the component parameters of the SMS service primitives. Each table consists of up to six columns, containing the name of the service parameter, a column each for the request (“Req”), indication (“Ind”), response (“Rsp”), and the confirm (“Cnf”) primitives, and a column for conformance building block specification (“CCB”). The “Rsp” and “Cnf” columns are absent when the service is not a confirmed service.

#### **5.2.1.    COMPANION STANDARD SERVICE PARAMETERS**

Some SMS service procedures which allow extensions to be supplied by Companion Standards. Such service procedures allow the specification of extra parameters in either the request primitive or in the response (+) primitive or both. This is indicated in the tables by the appearance of “COMP” following the Argument” or the “Result(+)” entry of the service table respectively. Some service procedures allow the specification of additional parameters by the Companion Standards directly in an Argument list. Such occurrences will be indicated by the “COMP” in the columns describing the service primitives.

#### **5.2.2.    SERVICE TABLE STRUCTURE**

For those tables that require support of particular horizontal conformance building blocks, the required conformance building blocks are enumerated on the first line in the table. In the remainder of each table, one parameter (or part of it) is listed on each horizontal line. Under the appropriate service primitive specified in the vertical column:

- M - parameter is mandatory for the primitive
- U - parameter is a user option, and may not be provided depending on dynamic usage by the SMS-user
- C - parameter is conditional upon other parameters or the environment of the SMS-user
- (blank) parameter is never present
- COMP - parameter is for definition in SMS Companion Standards. Such parameters shall not be used other than as defined in a Companion Standard (and shall be omitted in the abstract syntax defined in this Standard). Annex C provides additional detail on the use of Companion Standards.

## SuperMOCA Space Messaging Service

- S - parameter is a selection from a collection of two or more possible parameters. The parameters that make up this collection are indicated in the table as follows:
- a) each parameter in the collection is specified with the code “S”;
  - b) the name of each parameter in the collection is at the same indentation from the beginning of the parameter column in the table;
  - c) Either
    - 1) each parameter is at the leftmost (outer) indentation in the table; or
    - 2) each parameter is part of the same parameter group. A parameter group is a collection of parameters where each group member has a common parent parameter. The parent parameter for any group member is the first parameter above the member that is not indented as far as that member. In the example below, ParameterA and ParameterB form a parameter group:

```
ParameterX
  ParameterA
  ParameterB
ParameterY
  ParameterC
```

Informally, for parameters involved in a selection, the indentation in the services tables signifies which parameters are involved in a selection. All parameters at the same level of indentation that are under a common “higher level” parameter are part of the same selection.

The code “(=)” following one of the codes M, U, C, or S indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. (For instance, an “M(=)” code in the indication service primitive column and an “M” in the request service primitive column means that the parameter in the indication primitive is semantically equivalent to that in the request primitive.)

Some parameter may contain sub-parameters. Sub-parameters are indicated by labeling of the parameter as M, U or C, and indenting all sub-parameters under the parameter. Presence of sub-parameters is always dependent on presence of the parameter that they appear under (for example, an optional parameter may have sub-parameters; if the parameter is not supplied, then no sub-parameters may be supplied).

The CBB column is used to indicate that usage of the parameter is dependent on support of conformance building blocks, other than that containing the service. If no entry exists in the CBB column, then there is no dependency on other conformance building blocks. If an entry does exist, then the parameter is available (and permitted for use by this Standard ) if and only if the named conformance building block is supported and negotiated for use.

Some service parameters are named using a “List Of ...” convention. Unless otherwise noted, all parameters whose names begin with “List Of ...” specify a list of zero or more of the item specified after the “List Of...” keyword phrase. (This type of parameter corresponds with the sequence-of ASN.1 type in part 2 of this Standard.)

The descriptions of parameters in this part of this Standard make reference to types, in order to describe the allowable values for such parameters. The types referenced may either be types defined in ISO 8824 (Abstract Syntax Notation One), or may be defined in part 2 of this Standard.

### **5.3. INVOCATION IDENTIFIER ON SERVICE PRIMITIVES**

For services identified in the `ConfirmedServiceRequest` production in 7.5.2 of part 2 of the Standard, each SMS service primitive contains an “Invoke ID” parameter, which is mandatory in the request, indication, response, and confirm primitives. The value in the indication, response, and confirm primitives is semantically equivalent to that in the request primitive. This parameter serves to identify unambiguously the service invocation from an SMS-user on an application association. This parameter is not explicitly shown in the service primitive tables, nor is it explained separately for each service.

### **5.4. LIST OF MODIFIER ON SERVICE PRIMITIVES**

Every confirmed MMS service contains a “List Of Modifier” Parameters which is a user option in the request and indication primitives. The value in the indication primitive is semantically equivalent to that the request primitive. This parameter serves to specify a list of one or more service state modifiers which add a condition which must be satisfied for the execution of the service request to begin. This parameter is not explicitly shown in the service primitive tables, nor is it explained separately for each service.

SMS defines two modifiers: the `AttachToSemaphore` modifier and the `AttachToEventCondition` modifier, which are described in section 13 and 15, respectively.

The effect of the modifier on the state machine for execution of a confirmed SMS service is described in section 7.

### **5.5. ADDRESSING IN SMS**

The SMS Standard does not provide the means for naming and addressing of a peer SMS-user or peer Space Messaging Protocol Machine (SMPM). Within Open System, the identification and addressing of peer application entities is carried out through the use of ACSE services defined in ISO 8649. After such association of peers has succeeded, all of the SMS PDUs flow between these peers over the established presentation connection. It is therefore not necessary for SMS to carry addressing information. Additional information on naming and addressing may be found in section 6.6.

## **5.6. SERVICE CONVENTIONS**

This Standard uses the descriptive conventions contained in the OSI Service Conventions (ISO TR 8509). The OSI Service Conventions define the interactions between the SMS-user and the SMS-provider. Information is passed between the between SMS-user and the SMS-provider by service primitives, which may convey parameters. The following apply to the use of this model:

- a) ISO TR 8509 defines a model for the service provided by a layer of the OSI Reference Model. The SMS service does not correspond to such a layer (it describes a part of the application layer) but the model used is identical in all respects.
- b) At any instant in time, an application entity has multiple service requests outstanding, each processing independently of the others.

### **NOTE**

It should be noted that the SMS-user/SMS-provider distinction is an abstraction, and may not necessarily correspond to the realization of SMS in any particular system.

## **5.7. CALLING AND CALLED SMS-USER**

This Standard makes use of the terms Calling and Called SMS-users. The Calling SMS-user is the SMS-user that issues the Initiate.request service primitive. The Called SMS-user is the SMS-user that issues the Initiate.response service primitive.

### **NOTE**

The use of the term “called” in SMS is not the same as the general usage of the term in OSI. The SMS usage of the term “called” corresponds to the OSI usage of the term “responding”. This distinction has been introduced in order to avoid confusion with the Requesting/Responding SMS-user definition given below.

## **5.8. SENDING AND RECEIVING SMS-USER AND SMPM**

This Standard makes use of the terms Sending and Receiving SMS-users. The Sending SMS-user is the SMS-user that issues a request or response service primitive. The Receiving SMS-user is the SMS-user that receives an indication or confirmation service primitive.

### **NOTE**

It is important to note that, in the course of completion of a confirmed SMS service, both SMS-users will be senders and receivers at one time. The first SMS-user sends the request and receives the confirmation, while the second SMS-user receives the indication and sends the response.

This Standard makes use of the terms Sending and Receiving SMPMs. The Sending SMPM is the SMPM that sends an SMS PDU. The Receiving SMPM is the SMPM that receives an SMS PDU.

## 5.9. REQUESTING AND RESPONDING SMS-USER

This Standard makes use of the terms Requesting and Responding SMS-users. The Requesting SMS-user is the SMS-user that issues a request service primitive for a service, while the Responding SMS-user is the SMS-user that issues the response service for a service.

### NOTE

It is important to note that the use of the term Responding SMS-user differs from the use of the term Responding entity in ACSE and other Standards. In those Standards, the term is used to reference the entity that responds to a connection request.

## 5.10. CLIENT AND SERVER OF A SERVICE

This Standard makes use of the terms Client and Server in order to describe the model of the SMS VSD (the VSD is described in section 7). The Server is defined as the peer communicating entity which behaves as a VSD for a particular service request instance. The Client is the peer communicating entity which makes use of the VSD for some particular purpose via a service request instance. The VSD model is primarily useful in describing the actions of the Server, and thus in describing the commands and responses that a Client may use. A real end system may adopt the Client role, or the Server role, or both during the lifetime of an application association. Use of SMS in the OSI environment is further described in section 6.

Figure 5-1 depicts the relationships of the Client and Server of a service, the requesting and responding SMS-user, and the sending and receiving SMS-user and SMPM.

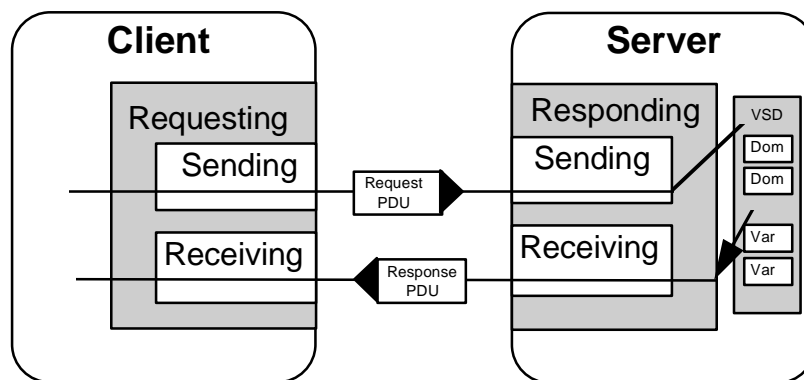


Figure 5-1 - Client-Server Relationships

## 5.11. OBJECT MODELING

This Standard makes use of a technique of abstract object modeling in order to fully describe the SMS device model and the SMS service procedures. In this modeling technique, abstract objects, the characteristics of such objects, and operations on those objects are described. The objects defined are abstract and aid in the understanding of the intent of SMS service procedures and their effects. In implementing SMS, a real system maps the concepts described in the model to the real device. Hence, as viewed externally, a device that conforms to this Standard exhibits the characteristics described in the object modeling technique, but the mechanisms for realization of this view are not defined by this Standard.

SMS defines a number of classes of objects. Each object is an instance of a class, and constitutes an abstract entity which exhibits certain characteristics and may be affected by certain SMS services and operations. Each class is given a name, by which it may be referenced.

Each class is characterized by a number of attribute types, which serve to describe some externally visible feature(s) of all objects of this class. Each instance of a class (object) has the same set of attribute types, but has its own set of attribute values. The values of these attributes are defined by this Standard or may be established by SMS services, and hence their effect on the device may be modeled by a change in one or more attribute values of an object (or objects).

Each object must be uniquely identified among all instances of the same class. For this purpose, one or more of the object's attribute values, as a combination, must be unique. (For example, many objects have an attribute type called "object name", which is different for each object of the same class.) In SMS, each attribute which is a part of this combination of attributes which make the object unique is identified as a "key attribute".

Finally, some objects contain attributes which are conditional, in the sense that they are relevant to the object if and only if certain conditions hold true. SMS expresses such attributes through the use of a "constraint", which specifies a condition. Attributes that are subject to a constraint are considered to be object attributes for an object if and only if the corresponding constraint is satisfied for that object.

In SMS, classes are syntactically defined as a set of objects as follows:

Object: (name of class)

Key Attribute: (name of attribute type (values))

.  
.  
.

Key Attribute: (name of attribute type (values))

Attribute: (name of attribute type (values))

.

```
.  
.
Attribute: (name of attribute type (values))
Constraint: (constraint expression)
    Attribute: (name of attribute type (values))
    Attribute: (name of attribute type (values))
    Attribute: (name of attribute type (values))
```

By convention, each object definition begins with an object declaration and the name of the object. Immediately following, and indented, one or more Key Attributes are named. Next, zero or more Attributes are named. Note that constraints may be expressed anywhere within the attributes, with the convention that all attributes subject to the constraint are indented underneath it. The first attribute definition that is not indented ends the list of attributes that are subject to the constraint.

#### NOTE

For convenience, attributes may be indented under other attributes in order to show a hierarchy of nesting of such attributes.

### 5.12. REFERENCE TO OBJECTS

Some objects contain attributes which make reference to other objects. Such attributes, called reference attributes, provide a mechanism to create a linkage from one object to another. The values used to represent such attributes in a real system is a local matter, and such attributes may not be directly modified or examined. Many SMS services provide the capability to determine the identity of an object referenced by such a linkage, however, through the use of such an indirect reference.

#### NOTE

Reference attributes are similar to “pointer” types available in many programming languages. As an example of the operation of reference attributes, the SMS Rename service may be used to change the Object Name attribute of a referenced object, while having no effect on the value of the reference attribute.

Reference attributes may take on the value UNDEFINED when the object referenced by the link is deleted. In this case, the consequences of this value and its use on the behavior on other objects is specified in the relevant object and service descriptions. Once a reference attribute become UNDEFINED, creation of an object of the same type as the deleted (and referenced object) with the same attribute values as that object which was deleted does not change the reference attribute (it retains the value UNDEFINED).

### 5.13. PARAMETER TYPES

Types for parameters defined in this part of this Standard make use of types defined in section 3 of ISO 8824. Additionally, complex types are used, which are constructed from the ISO 8824 primitive types and subsequently named to allow them to be referenced.

## SuperMOCA Space Messaging Service

## 6.     **THE SMS ENVIRONMENT**

This section serves to describe the relation between SMS and the OSI environment, and relates OSI terms and concepts to the space environment. Section 7 describes the specific model of the SMS device within this environment.

The development of standards for the interconnection of spacecraft monitoring and control devices is assisted by the use of abstract models. To specify the externally perceived behavior of interconnected spacecraft devices, each spacecraft device is represented by a functionally equivalent abstract model of the device called a Virtual Spacecraft Device (VSD) (see section 7). The VSD model, along with the extensions to this model which are provided by sections 8 to 16, describes the externally visible aspects of these devices.

In order to accomplish this, it is necessary to describe both the internal and external behavior of these spacecraft devices. Only the external behavior of the devices is retained as the standard of behavior of a VSD. The description of the internal behavior of such devices is provided in the model only to support the definition of the externally perceived aspects. Any spacecraft device which behaves externally as a VSD can be considered to in conformance with this Standard.

### NOTE 1

The reader not familiar with the technique of abstract modeling is cautioned that the concepts introduced in the description of the VSD constitute an abstraction despite a similar appearance to concepts commonly found in real devices. Therefore, the VSD model is not a specification for an implementation.

Additional information to supplement this section may be found in the OSI Reference Model (ISO 7498-1 and ISO 7498-3), the OSI Application Layer Structure (ISO 9545), and the Service Definition for the Association Control Service Element (ISO 8649). References may be found in section 2 of this part of the Standard. This information may be particularly useful in understanding the relationship between AEs, APs, and their respective invocations.

### NOTE 2

This Standard specifies SMS operation in the OSI environment only. Operation in other environments and architectures is beyond the scope of this Standard, but may be standardized elsewhere.

## **6.1.   INFORMATION PROCESSING TASKS AND REAL SYSTEMS**

The control and monitoring of a spacecraft process is a distributed information processing task. In order to successfully carry out this task, inter-operation of a number

of real open systems is required. In OSI, a real open system is defined as a set of computers and associated software (including peripherals, terminals, human operators, physical processes, etc.) that complies with the requirements of OSI standards in its communications with other real systems.

In OSI, the inter-operation of real systems is modeled in terms of the interactions between Application Processes (APs) in these systems. The distributed task of control and monitoring of a spacecraft process requires the co-operation of two or more Application Processes.

### **6.2. APPLICATION PROCESSES**

An Application Process (AP) is an element within a real open system which takes part in the execution of one or more information processing tasks. It is an abstract representation of those aspects of a real open system which are specific to the performance of those tasks. APs generally have requirements above and beyond any requirement to communicate with other APs. In the spacecraft environment, an AP represents a real system participating in the overall control/monitoring distributed task.

### **6.3. INTERACTION OF APPLICATION PROCESSES**

Several requirements must be met in order to allow cooperative operation of APs. First, they must share sufficient information to enable them to interact and carry out processing functions in a compatible and cooperative manner. In the spacecraft environment, the term “universe of discourse” is used to name a model of those aspects of the “world” which are pertinent to the objectives of a spacecraft control and monitoring task of an AP. In order to allow successful interworking between APs, they must share a universe of discourse (i.e. they must have a common understanding of the spacecraft environment).

It is convenient to describe the common model of the spacecraft environment, or universe of discourse, in terms of a set of abstract “objects”. Examples of objects in the spacecraft environment include variables, programs, and semaphores. The model describes the characteristics of objects and relationships between them. The characteristics may include the properties of those objects (either static or dynamic), and these properties are expressed as a set of rules and constraints about the behavior of these objects within the model of the spacecraft environment. An example of a property of an object is the characteristic of a program that it is either running or stopped.

A universe of discourse can be described formally by a “conceptual schema”. The conceptual schema for the spacecraft universe of discourse is described by the models provided by SMS. The second requirement for successful interworking of two APs is that they share a common model of the objects (such as variables, programs, and semaphores) in the universe of discourse. In OSI, this is called a “shared conceptual schema”. The requirement is met in a spacecraft control and monitoring application through the sharing of the models in SMS for the spacecraft universe of discourse.

When two APs cooperate, their behavior is determined partly by these models (the share conceptual schema) and partly by their past interactions. The past integrations are modeled by the state of the objects in the universe of discourse. As an example, the cooperative behavior of two APs may be partly determined by the Program Invocation model of SMS and partly by the state of Program Invocation objects in the APs. The shared information about the state of objects is called an information base.

### **6.4. INTERACTION OF APPLICATION PROCESSES IN SPACE**

The activity of a given AP in a specific Information Processing task is supported by an application-process-invocation. At any time, an Application process may have zero, one or more application-process-invocations. While the Application Process describes a specific set of Information Processing functions in a particular real system, the application-process-invocation (AP-invocation) describes an instance of the Application Process in a real system for a particular occasion of Information Processing. (Hence, an AP may describe control of an instrument in a specific real system on some occasion of control for data acquisition).

Thus, co-operation between APs for the performance of a given Information Processing task takes place through interacting AP-invocations. When APs interact using OSI communication capabilities, these communication capabilities are in turn modeled by one or more Application Entities (AE). An AE is an active element in the Application Layer of an open system, and it represents those parts of an Application Process involved in OSI communications. Each AE represents exactly one AP.

Like APs, the activity of a given AE is supported by an application-entity-invocation. An AE-invocation performs the functions of an AE for a particular occasion of communication. At any time, an AP-invocation may be represented by zero or more AE-invocations for each AE associated with the AP. Hence, the interactions of AP-invocations for a particular occasion of communication is represented in OSI by a set of corresponding AE-invocations.

Because an AE describes only part of the operation of an AP (namely, that involving OSI communications), resources in AP-invocations may exist beyond the lifetime of (and may be independent of) an AE-invocation.

#### **NOTE**

The relationship between an AE and the VSD is defined in section 7.

### **6.5. STRUCTURE OF APPLICATION ENTITIES**

The AE represents a set of communication capabilities of an AP. These capabilities are defined by a set of Application Service Elements (ASEs). An ASE is a coherent set of integrated functions that provides a capability for communications for some particular purpose. SMS is modeled as an ASE for the purpose of communications with spacecraft devices. Support for the Association Control Service

Element (ACSE) ASE, as defined in ISO 8649, is required in every AE that supports the SMS ASE.

## **6.6. ADDRESSING OF APPLICATION ENTITIES**

Part 3 of ISO 7498 describes naming and addressing concepts in the OSI environment. This describes how some of these concepts apply to SMS.

A Presentation-address is associated with a set of PSAPs in a single real system. An E is attached to one or more Presentation Service Access Points (PSAPs) in order to make it addressable in the OSI environment.

Each AE is identified by one or more application-entity-titles (AE-titles), which are unambiguous throughout the OSI environment (OSIE). At any instant of time, each AE-title is bound to a single Presentation-address that identifies the set of PSAPs to which the AE is attached. This binding is recorded in the Application Title Directory Service, which contains information about AEs.

A system may access the Application Title Directory Service via OSI Directory Service (ISO 9594). Conceptually, each system contains a directory function, from which AEs may obtain addressing information. This directory function may operate via the use of a local cache, or may utilize the Application Title Directory Service via the OSI Directory Service, in order to satisfy requests from AEs.

### **NOTE**

This is only a conceptual model, and that there are many valid implementation styles.

In order for an application-entity to establish an application association (AA) with another AE, it must know the Presentation-address or obtain it by using the AE-title to get the Presentation-address of the AE from the directory function. It then uses this Presentation-address to establish a presentation connection with the called AE. The mechanisms for control of associations are provided in ISO 8649 and ISO 8650 by the Association Control Service Element (ACSE). Since an AE is the representation of an AP within OSI, the establishment of communication to support an association with the called AE is the establishment of communication with the related AP.

SMS makes use of an “Application Reference” to identify an AE in another system, by making use of its title as defined in ISO 8649. The syntax of an Application Reference is defined in part 2 of this Standard.

## **6.7. APPLICATION CONTEXT**

The application context, which is established via the Association Control Service Elements (ACSE) (ISO 8649), identifies the set of application service elements, their options, rules for use of the service elements, and their effects that are available on an application association. In the case of SMS, requirements associated with this Standard are in effect when the SMS application context is negotiated.

## **6.8. PRESENTATION CONTEXT, ABSTRACT SYNTAXES, AND TRANSFER SYNTAXES**

In OSI, there is an important distinction between the generic requirements of an application for the transfer of data and how those requirements are met in terms of a specific representation of data values. The former aspect of data description is referred to by the term “abstract syntax”, while the latter aspect is referred to by the term “transfer syntax”.

Abstract syntaxes are intimately associated with application protocol standards. Part 2 of this standard defines an abstract syntax matching its data transfer requirements. An abstract syntax can be viewed informally as describing the generic structure of data. In SMS, the set of type definitions provided in part 2 of this standard constitutes an abstract syntax. The SMS abstract syntax may be supported by many different transfer syntaxes.

Transfer syntaxes are concerned with the way in which data is actually represented in terms of bit patterns during transmission. A transfer syntax may have attributes that are not related to the abstract syntaxes which it can support, but such attributes may be significant. For example, a transfer syntax may provide data compression or encryption. Such attributes (and how well they meet the requirements of the device) may influence the choice of syntaxes offered or selected for an instance of communications.

Abstract Syntax Notation One (ASN.1) is an example of a tool for specification of syntaxes and associated encoding rules. (The application of encoding rules to a particular abstract syntax may generate a transfer syntax.) The ASN.1 Specification (ISO 8824) has been chosen to describe the SMS abstract syntax. One possible transfer syntax for SMS is defined by the application of the ASN.1 Encoding Rules (ISO 8825) to the SMS abstract syntax. However, the limitations and demands on spacecraft hardware and communication resources makes it desirable to minimize the overhead burden of encoding. Therefore, this standard requires that any system claiming conformance to this standard shall support the transfer syntax that results from the application of the Packed Encoding Rules to the SMS abstract syntax. Support of other transfer syntaxes is a local matter.

[Note: the selection of Packed Encoding Rules for SMS is under investigation. There are several other encoding solutions are also under investigation including a new scheme that has been proposed by a team at the Swiss Federal Institute of Technology called Tagless Encoding Rules (TASN). A final recommendation will be made prior to the formal submission of this standard as a full standard. See Special Attachment 1 for addition information on TASN. ]

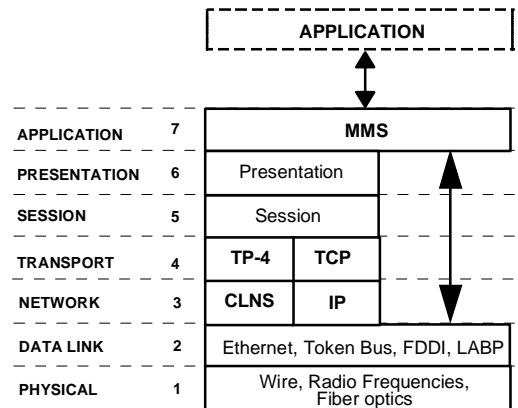
## **6.9. LOWER LAYER PROTOCOL STACK CONSIDERATIONS**

[ New section specific to SMS ]

[Note: The following discussion addresses issues regarding the communications environment required to support SMS. The discussion is intended to clarify the dependencies and non-dependencies of SMS on lower layer protocol issues. It is also intended to clarify some expected implementation issues that intrude into application layer

entity discussions. Whether or not this discussion should appear here, somewhere else in the document, or not at all may also be a topic for discussion. ]

The SMS is a derivative of the Manufacturing Message Specification (MMS) standard (ISO 9506) developed for factory automation and designed originally to operate over a full seven layer OSI protocol stack. As an application layer entity, SMS like MMS has limited dependencies on an underlining communications architecture. MMS has been successfully used over a full 7 layer OSI protocol stack ( using TP4, CLNS ) operating over a variety of data link protocols (ethernet, token bus, FDDI, HDLC, LAPB and serial communications). MMS has also been successfully used over an Internet protocol stack ( using TCP/IP ) based on RFC 1006. In addition, MMS has been implemented in an Enhanced Performance Architecture (EPA) where the application layer entity operates directly over the data link layer ( token bus, LAPB, HDLC ). Figure 6-1 illustrated the variety of environments in which MMS has been implemented.

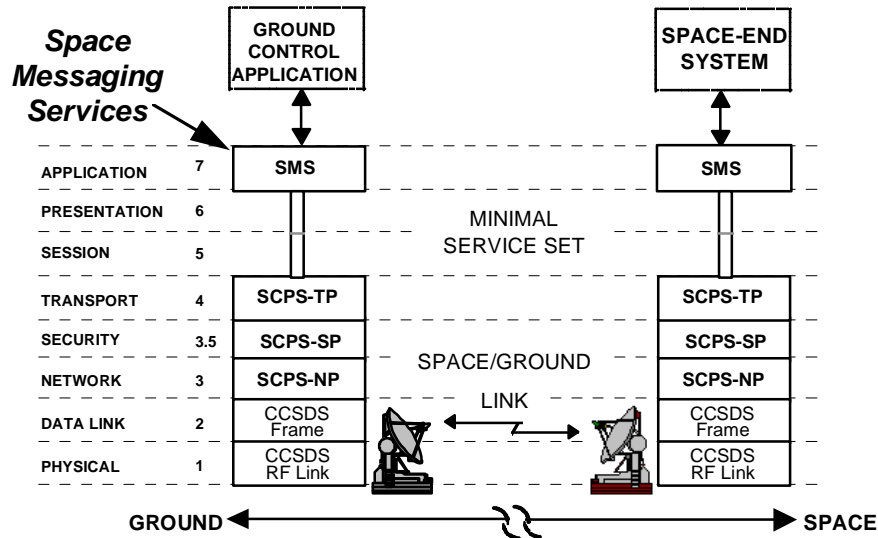


**Figure 6-1 - Implementation environments in which MMS has been successfully deployed**

## 6.9.1. FULL STACK

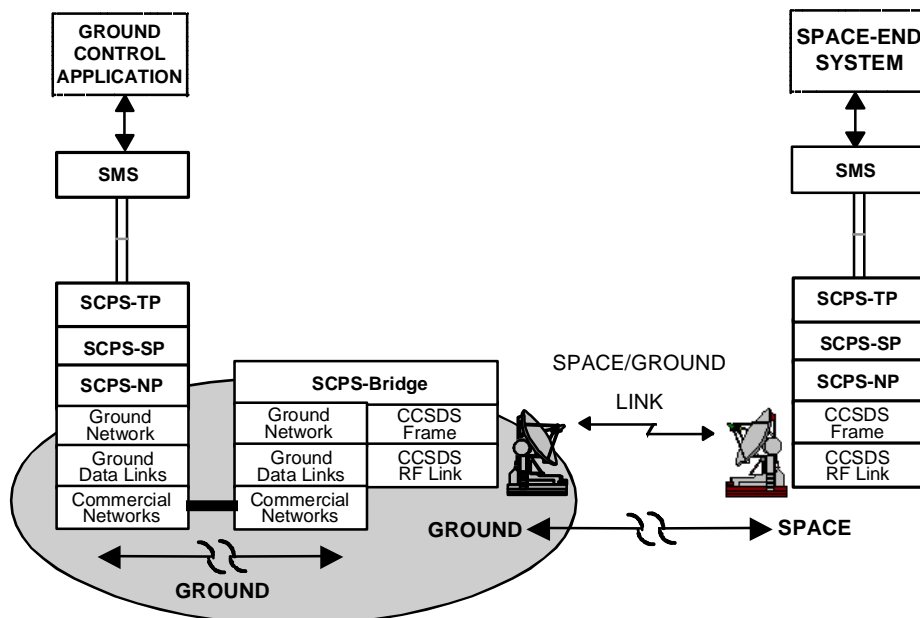
The SMS shall use the Space Communications Protocol Standards (SCPS) for ground-to-spacecraft, spacecraft-to-ground and spacecraft-to-spacecraft. The SMS with employ a minimal service set of OSI presentation and session layer services above the SCPS stack to support the exchange of messages. Figure 6-2 illustrates the peer-to-peer relationship for the exchange of SMS messages between applications.

## SuperMOCA Space Messaging Service



**Figure 6-2 - The peer-to-peer relation for Ground-to-Spacecraft, Spacecraft-to-Ground and Spacecraft-to-Spacecraft is shown above for SMS operating over the SCPS stack.**

While the model shows the peer-to-peer communications path, communications in real systems will likely employ communication gateways to bridge ground communication networks through Space Link Extension facilities to support ground-to-spacecraft and spacecraft-to-ground messaging. Figure 6-3 illustrates the communications path and communication layers employed to support the exchange of data.



**Figure 6-3 - Communications Paths and Layers**

In practice, commercial ground networks will be employed to support communications between mission operation centers and Space Link Extension facilities

(i.e. tracking stations). The peer-to-peer relation for Ground-to-Spacecraft, Spacecraft-to-Ground and Spacecraft-to-Spacecraft will employ gateways to support the transition to the space communications environment.

### 6.9.2. ENHANCED PERFORMANCE ARCHITECTURE

As an application layer entity, SMS has limited dependencies on an underlining communications architecture. The application of SMS for onboard systems is open to employ a SCPS stack or an Enhanced Performance Architecture (EPA) stack (no transport or network layers). The EPA approach is open to employ a variety of data link layer protocols that might provide for error detection and retransmission ( Logical Link Control 2 - LLC2 for example ). In addition, the design and operational requirements of the onboard devices may impose specific requirements for the selection of the data link communication services. Figure 6-4 illustrates the peer-to-peer messaging in the EPA environment.

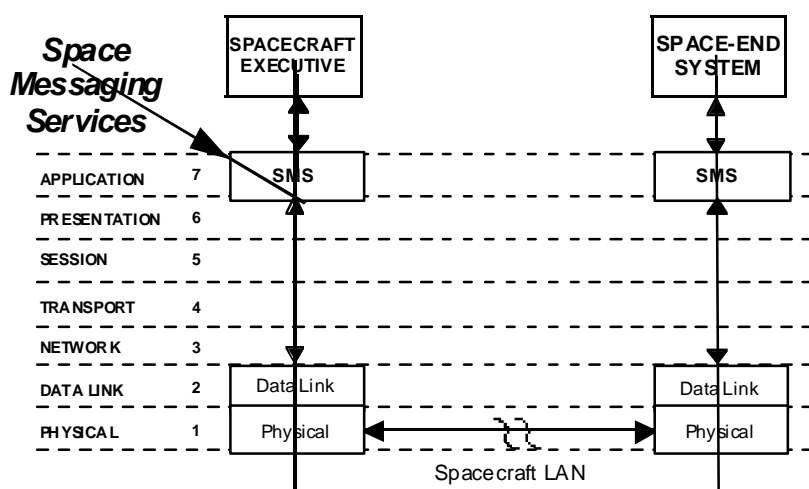
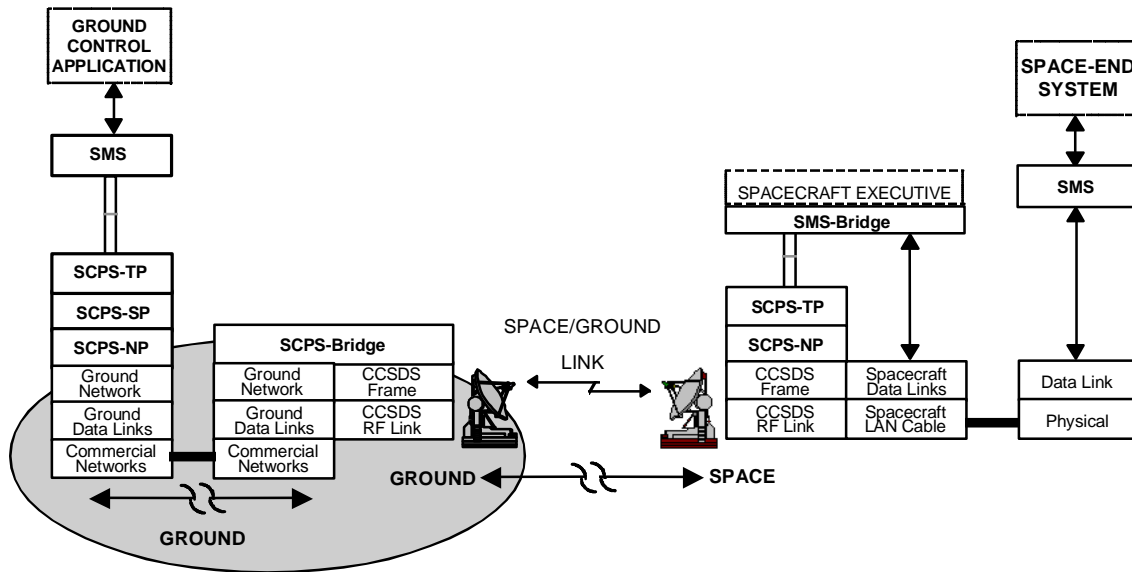


Figure 6-4 - The peer-to-peer relation for onboard systems using SMS over the EPA stack

[Note: For example: If the onboard devices are designed with internal closed-loop control on all critical elements, a non-deterministic data link such as ethernet could be used to support inter-device communications. If onboard devices require highly deterministic access to communication services to support closed-loop control, a token-passing data link could be used to support inter-device communications. Further, a spacecraft such the Space Station might employ multiple LANs. In such cases, there might be LAN segments that employ SMS over SCPS and LAN segments that employ SMS over data link. In these environments, the segments might be linked together using repeaters and bridges. ]

While the model shows the peer-to-peer communications path, communications in real systems will employ an SMS bridge to route messages based on their AP-Title (which is derived from the AP's PSAP). Figure 6-5 illustrates the communications path and

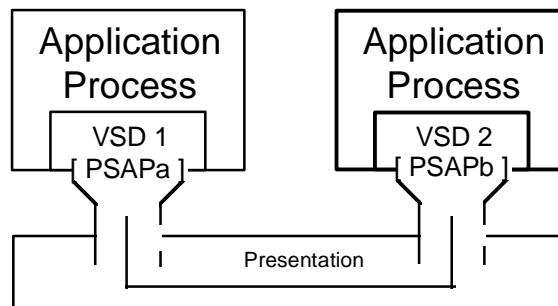
communication layers employed to support the exchange of data between a ground application and a space-end system ( which might be an instrument or sub-system ).



**Figure 6-5 - The peer-to-peer relation for a ground system communicating with an onboard systems using SMS over the EPA stack**

### 6.9.3. INTER-TASK COMMUNICATIONS

Inter-task communications is supported by SMS based on the AP-Title. Application processes employ the AP-Title to direct messages to local or remote applications without regard for transport medium (see Section 7.1.1). Figure 6-6 illustrates inter-task communications between application processes.



**Figure 6-6 - The peer-to-peer relation for inter-task communications**

## **7. THE VIRTUAL SPACECRAFT DEVICE MODEL**

[Note: The MMS standard was developed using an abstract modeling technique that defined a Virtual Manufacturing Device (VMD). The VMD represents the externally visible attitudes of a real manufacturing device. The services required to manage a VMD are specified in MMS. From the SuperMOCA perspective, spacecraft subsystems, devices and instruments can be modeled as Virtual Spacecraft Devices (VSDs). The services required to manage a VSD are specified in SMS. ]

### **7.1. INTRODUCTION**

The Space Messaging Service defines the externally visible behavior of an SMS server application process. This behavior is modeled by describing an entity called a Virtual Spacecraft Device (VSD). This section describes the model for a VSD. It explains the VSD's relationship to the application process and the application entity (AE). It defines the structural elements of the VSD and introduces the abstract objects which exist at, and are manipulated by, a VSD on behalf of a client SMS-user.

An implementation of an SMS server must provide a mapping of the VSD model to the functionality of a real spacecraft device. Guidance in the selection of a particular mapping may be found in various Companion Standards. These Companion Standards address the specific needs of discrete parts of spacecraft systems - power subsystems, attitude control subsystems, navigation subsystems, imaging instrument systems, spectrographic instrument systems, etc.

#### **NOTE**

The SMS services do not constrain the behavior of a client SMS application process, except with respect to valid sequences of primitives. Therefore a model of the SMS client application process is not given.

#### **7.1.1. RELATIONSHIP OF THE VSD TO THE OSI MODEL**

A VSD exists within the SMS server application process. It constitutes that portion of an information processing task which makes available - for control, or monitoring, or both - a set of resources and functionality associated with a real spacecraft device. An application process may contain zero or more VSDs. If it does not define a VSD it may not act as an SMS-server.

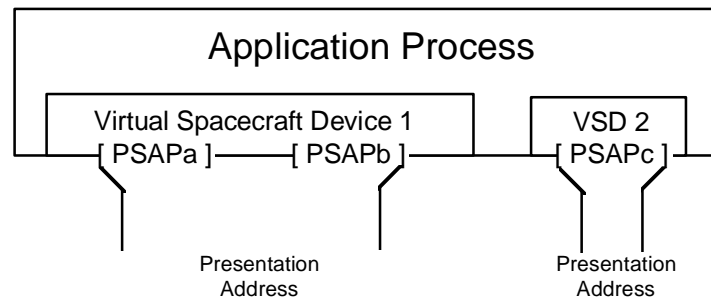
Each VSD represents a virtual device within that application process (AP), and each VSD is logically separate from all other VSDs. Each AP may contain zero or more application entities (AEs), such that an AE represents a set of communication capabilities of the AP (an AP that contains no AEs may not communicate in the OSI environment).

A VSD may wholly contain zero or more AEs. Each AE in a VSD represents a set of communication capabilities used by the aspects of the AP represented by a VSD. Each AE is related to one and only one VSD. When a VSD contains more than one AE, then it contains more than one set of communication capabilities.

At any instant of time, each AE-title is bound to a single presentation-address that identifies a set of PSAPs to which the AE-title is bound and therefore the AE-title is addressable in OSI. Communication with a particular AE is used to model communication with a VSD, and hence an AP. Application associations are modeled as taking place between AEs (and hence VSDs and APs).

An AE within an AP may have zero or more AE-invocations. For the purposes of SMS, each AE-invocation models an instance of usage of that AE in an application association. Hence, several AE-invocations of an AE may be used to model several application associations with a VSD that contains that AE. By generalization, there may simultaneously exist several AEs of a VSD, with several associations each (modeled by multiple AE-invocations). Note that in OSI a more generalized view allows an AE-invocation to have more than one application association.

An OSI Presentation Address which identifies one or more PSAPs addresses a single VSD. This binding of a Presentation Address to a VSD is of a relatively long duration. Figure 7-1 illustrates the relationship between an OSI application process acting as an SMS server, its VSDs, and the PSAPs used to access them.



**Figure 7-1 - The SMS server Application Process**

### **7.1.2. RELATIONSHIP OF THE VSD TO THE REAL SPACECRAFT DEVICE**

A VSD is an abstract representation of a specific set of resources and functionality at the real spacecraft device and a mapping of this abstract representation to the physical and functional aspects of the real spacecraft device. This mapping of a virtual resource to the underlying actual resource is of relatively long duration.

Generally, the resources of a given VSD are distinct from, and independent of, the resources of all other VSDs. When a virtual resources of two (or more) VSDs is mapped to the same underlying physical resource, a mechanism must be provided by the application process(es), and made available through the VSDs, so that clients of the various VSDs may coordinate their access to the single real resource. This coordination may be modeled by requiring each VSD to obtain control of a “virtual” semaphore whenever disruptive access to the virtual resource is being requested by an SMS service. This virtual semaphore would then be mapped on to a real semaphore which controls access to the real resource. With this approach, the virtual semaphores of the various VSDs are independent when viewed using SMS services. In other words, users of the

virtual semaphore of one VSD appear as though part of the resident system when viewed by users of the semaphores of the other VSDs.

SMS describes the operation of a VSD by describing the abstract objects which are manipulated by it, and by describing the set of operations which may be performed on these objects through use of the SMS services.

## **7.2. THE STRUCTURE OF A VSD**

Each VSD contains exactly one Executive Function and zero or more Program Invocations, each of which depend on one or more Domains. The Domain represents a specific use of a set of capabilities of a VSD. The state of the VSD, in the complete sense, is determined by the values of all the attributes of the VSD, including all the attributes of its Domains and their subordinate objects. The elements of the VSD are described below.

The VSD Object Model

Object: VSD

Key Attribute: Executive Function

Attribute: Vendor Name

Attribute: Model Name

Attribute: Revision

Attribute: List of Abstract Syntaxes Supported

Attribute: Logical Status (STATE-CHANGES-ALLOWED, NO-STATE-CHANGES-ALLOWED, LIMITED-SERVICES-PERMITTED, SUPPORT-SERVICES-PERMITTED)

Attribute: List Of Capabilities

Attribute: Physical Status (OPERATIONAL, PARTIALLY-OPERATIONAL, INOPERABLE, NEEDS-COMMISSIONING)

Attribute: List Of Program Invocations

Attribute: List Of Domains

Attribute: List Of Transaction Objects

Attribute: List Of Upload State Machines (ULSM)

Attribute: List Of Other VSD-specific Objects

Attribute: Additional Detail

### **7.2.1. THE EXECUTIVE FUNCTION**

The existence of a fully functional Executive Function exactly corresponds with the existence of the VSD.

### **7.2.2. VENDOR NAME**

This attribute is a character string which identifies the vendor of the system which supports this VSD.

### **7.2.3. MODEL NAME**

This attribute is a character string which identifies the model of the system which supports this VSD. The value of this string is normally assigned by the vendor.

#### **7.2.4. REVISION**

This attribute is a character string which identifies the revision number of the system which supports this VSD. The value of this string is normally assigned by the vendor.

#### **7.2.5. LIST OF ABSTRACT SYNTAXES SUPPORTED**

This attribute identifies the set of Abstract Syntaxes which this VSD is able to support in the SMS Application Context. This set includes any abstract syntaxes defined by a Companion Standard as well as any abstract syntax which can be recognized as an encoding for Load Data (see 10.3) and Execution Argument (see 11.4). The abstract syntax defined in this Standard shall not be included in the list.

#### **7.2.6. LOGICAL STATUS**

There are four separate levels of functionality available through SMS which are described by the Logical Status attribute of the Executive Function.

##### **7.2.6.1. STATE-CHANGES-ALLOWED**

In this case, all SMS service which are supported by this VSD may be performed.

##### **7.2.6.2. NO-STATE-CHANGE-ALLOWED**

In this case, the only SMS services which may be performed (if the VSD supports the service) are:

- Abort
- Conclude
- Cancel
- GetAlarmEnrollmentSummary
- GetAlarmSummary
- GetCapabilitiesList
- GetDomainAttributes
- GetEventActionAttributes
- GetEventConditionAttributes
- GetEventEnrollmentAttributes
- GetNamedTypeAttributes
- GetNamedVariableListAttributes
- GetNamedList
- GetProgramInvocationAttributes
- GetScatteredAccessAttributes
- GetVariableAccessAttributes
- Identify
- Initiate
- Read
- ReadJournal
- ReportEventActionStatus
- ReportEventConditionsStatus

ReportEventEnrollmentStatus  
ReportJournalStatus  
ReportPoolSemaphoreStatus  
ReportSemaphoreEntryStatus  
ReportSemaphoreStatus  
Status

#### **7.2.6.3. LIMITED-SERVICES-PERMITTED**

In this case, the only SMS services which may be are: Abort, Conclude, Status and Identify.

#### **7.2.6.4. SUPPORT-SERVICES-ALLOWED**

In this case, all SMS services supported by the VSD except the Start, Stop, Reset, Resume and Kill services may be performed.

#### **7.2.7. LIST OF CAPABILITIES**

A VSD represents the real device to the client by providing a set if capabilities which may be used by the client in order to effect some control or monitoring (or both) activity through the VSD.

A capability is a locally defined resource (physical or logical) or a locally defined set of resources which can be identified by a character string. The definition and management of capabilities is outside the scope if this Standard. However, it is assumed that the existence and status of capabilities is known by the Executive Function, and that the Executive Function contains sufficient knowledge in order to “allocate” capabilities to the various Domains which may be created. No object model is provided for a capability, since it is considered primitive from the SMS point of view.

A capability may be sharable (or not) depending on the local criteria. Additionally, a capability may be distinct, or it may encompass (or be encompassed by) one or more other capabilities.

#### **NOTE**

An implementation of an SMS server should provide a mapping of the VSD model on to the functionality of a real spacecraft device.

#### **EXAMPLE 1**

In an attitude control subsystem, the real spacecraft device may have the capability of applying a thrust of +/- 0.002 Newtons in the X-axial direction, +/- 0.004 Newtons in the Y-axial direction and +/- 0.0038 Newtons in the Z-axial direction. The List Of Capabilities might reflect this capability in the following way: .....;Thrust\_X=+/- 0.002N,Y=+/-0.004N,Z=+/-0.0038N;.....

#### **EXAMPLE 2**

In an imaging instrument, the real spacecraft device may have a CCD camera capable of a 1024x1024 pixel image through a 50mm, F1.2 lens. The List Of Capabilities might reflect this capability in the following way:  
.....;Image\_50mmF1.2,1024(2);.....

### **7.2.8. PHYSICAL STATUS**

Associated with each capability is one or more attributes which describe the state of the capability. It is outside the scope of this Standard to provide a uniform representation of these attributes. However, it is useful to provide a standard representation of the gross aspects of all the capabilities, taken together, in order to characterize the operational state of the hardware. The Physical Status attribute (of the entire collection of capabilities) represents this attribute of the real device.

#### **7.2.8.1. OPERATIONAL**

The real device associated with this VSD has no known deficiencies and is able to perform its intended tasks.

#### **7.2.8.2. PARTIALLY-OPERATIONAL**

One or more functions of this real device cannot be performed due to hardware malfunctions or limitations.

#### **7.2.8.3. INOPERABLE**

One or more significant problems exist at the real device can prevent it from doing any useful task.

#### **7.2.8.4. NEEDS-COMMISSIONING**

The device is in a state such that a local commissioning process needs to be performed before useful tasks can be accomplished.

#### **NOTE**

Spacecraft designed to be launched and never serviced are unlikely to make use of devices built with this mode of operation possible.

### **7.2.9. LIST OF PROGRAM INVOCATIONS**

A Program Invocation consists of a set of procedural and data elements contained within Domains together with execution control information. These elements may be pre-defined within the VSD, or they may be dynamically defined (for example by creation of a Domain through use of the SMS load services), or both. The Program Invocation itself may be pre-defined within the VSD, or it may be dynamically created and deleted by either local means or through the use of SMS services. An object model of the Program Invocation is given in section 11.

#### **7.2.10. LIST OF DOMAINS**

A Domain represents a specific instance of use of a set of capabilities of the VSD. A Domain includes those aspects of a VSD which are associated with a specific element (possibly all) of a coordinated control or monitoring (or both) strategy. The allocation of the capabilities of a VSD to a Domains may be static or it may be dynamic. If the allocation is status, then the Domain is pre-defined within the SMS server and its name is considered to be known. If the allocation is dynamic, the Domain comes into existence and is removed from the VSD either through the action of SMS service or through local actions. Within a given VSD, either or both types of Domains may exist. Domains are further described and an object model provided in section 10.

A Domain may be empty or it may contain “information”. The “information” may be program instructions for some processor or tables of values or other classes of data or all of the above. For dynamic Domains which come into existence through SMS services, the Domains are implicitly created by the process of loading their contents. Dynamic Domains may also be created through actions of the Program Invocation when it is in execution or through other local means.

A Domain content is often closely associated with a file. The concept of file, although it appears in some SMS services, is not defined within SMS. The file may be part of a virtual Filestore which is part of the Application Process which contains the VSD, or it may be locally defined. If the “information” within a file is a part of the VSD and can be used directly by the VSD (as a component of a Program Invocation), it can be considered to be a Domain. The file, however, must be considered a separate object, outside the scope of the VSD.

Most SMS objects are defined subordinate to (within the name space of) a Domain. Thus, a Domain represents a single name space in which SMS objects must be uniquely identifiable.

#### **7.2.11. TRANSACTION OBJECT**

Most SMS services are confirmed services (see section 6 of part 2 of this Standard). When the VSD receives an indication primitive for one of the confirmed services, a Transaction Object is created which governs the processing of this service. The description of that object follows:

Object: Transaction

Key Attribute: Invoke ID

Key Attribute: Application Association Identifier

Attribute: List Of Pre-execution Modifiers

Attribute: Current Modifier References

Attribute: Confirmed Service Request

Attribute: List Of Post-execution Modifiers

Attribute: Cancelable (TRUE, FALSE)

## SuperMOCA Space Messaging Service

### Invoke ID

This attribute is an integer which serves to identify the transaction within the Application Association.

### Application Association Identifier

This attribute identifies the application association over which this transaction object is created.

### List Of Pre-execution Modifiers

An ordered list which identifies modifiers which must be satisfied before execution of the Confirmed Service Request attribute can begin.

### Current Modifier References

References the Semaphore Entry object or the Event Enrollment object controlling the current modifier's execution.

### Confirmed Service Request

Service identifier and Argument of the pending service.

### List Of Post-execution Modifiers

References the Semaphore Entry objects which this service invocation owns due to a processed Attach to Semaphore modifier.

### Cancelable (TRUE, FALSE)

Initially true, the service may set this attribute false (local issues). When true, cancel works, when false, cancel does not work.

### **7.2.11.1.Initialization of Transaction Objects**

A transaction object shall be created upon receipt of an indication service primitive for an SMS confirmed service, and deleted after the SMS-user issues a response service primitive for that service instance. The number of transaction objects which may exist at any time is governed by the negotiated maximum number of services outstanding (see 8.2).

When an indication service primitive is received, the List Of Pre-execution Modifiers attribute of the newly created transaction object shall be set based on the List Of Modifier parameter on the indication service primitive (see 5.4). The Current Modifier Reference is set to one, indicating the first Modifier on the List of Pre-execution Modifiers, or to zero if there are no Pre-execution Modifiers. the List Of Post-execution Modifiers is set to empty (no modifiers are on this list).

The Confirmed Service Request attribute of the transaction object shall be set based on the Argument parameter received on the indication service primitive.

The Cancelable attribute shall initially be set to true. The Cancelable attribute of the transaction object shall not be set to false during the processing of the pre-execution modifiers. After all the pre-execution modifiers have been processed, the cancelable attribute may be set to false by the SMS-user at any time as a local matter, subject to the requirements for the Cancel service and the service named in the Confirmed Service Request. The cancelable attribute shall have been set to false by the time that post-execution modifier processing takes place.

### **7.2.11.2.Processing of Transaction Objects**

After a transaction object has been initialized by the SMS-user, processing may begin on that object. Processing begins with the List Of Pre-execution Modifiers, followed by execution of the Confirmed Service Request, and terminates with processing of the List Of Post-execution modifiers.

Each of the Pre-execution modifiers shall be processed in order. Each modifier shall complete successfully before the next modifier can be processed. If a modifier fails (see definition of particular modifiers), the SMS-user shall process the Post-execution modifier list (as specified below) and then issues a response (-) service primitive, specifying the appropriate error class and code, and the transaction object shall be deleted. The Current Modifier Reference shall be set to indicate the modifier currently being processed in the list. The Current Modifier Reference shall be identified by an integer, where 1 indicates the first element in the list. For each AttachToSemaphore modifier which is successfully processed, an entry shall be made in the List Of Post-execution Modifier attribute (in order to allow relinquishing of control of the semaphores after service execution). The List Of Post-execution Modifier shall be constructed in reverse order, such that the first pre-execution modifier becomes the last post-execution modifier.

After all pre-execution modifiers have been processed successfully (if any are specified on the indication service primitive), the Confirmed Service Request shall be executed in accordance with the service procedure specified in this Standard for the named service.

Following completion of the service request, the SMS-user shall process the post-modifiers in the List Of Post-execution Modifiers attribute in the order that they are specified in the list (note that this is the opposite of the order in which the pre-modifiers were executed).

After all post-execution modifiers have been processed, the response service primitive shall be issued by the SMS-user. The Current Modifier Reference attribute is used to indicate the current modifier during the processing of this list.

### **7.2.12. UPLOAD STATE MACHINES**

The Upload State Machine object (ULSM) is created through the InitiateUploadSequence service which is described in section 10.

### **7.2.13. OTHER VMD-SPECIFIC NAMED OBJECTS**

There are many other named objects which may have VSD-specific scope. The attributes of such objects shall be considered part of the VSD. General considerations for all named objects are described in 7.3, and the objects themselves are described in separate sections of this Standard.

### **7.2.14. ADDITIONAL DETAIL**

This attribute contains zero or more attributes whose meaning and syntax are defined by appropriate Companion Standards.

## **7.3. SPECIFICATION OF NAMED OBJECTS**

SMS objects are normally referenced by name. The name of an object shall be unique within its scope of definition and within the class of object it identifies. In SMS, the names of objects are defined within one of three scopes: within a VSD instance, within a Domain, or within a single application association.

### **7.3.1. SCOPE OF NAMES**

In general, SMS names can have one of three scopes, VSD-specific, Domain-specific, and Application Association-specific.

#### **7.3.1.1. VSD-specific Scope**

A name which has VSD-specific scope shall be unique among all VSD-specific objects of the same object class. Such a name may be referenced by all clients of the VSD instance on any application association with it. VSD-specific objects are not automatically deleted when the SMS application association ceases to exist.

#### **7.3.1.2. Domain-specific Scope**

A Domain represents a single flat name space. A name which is Domain-specific shall be unique for its class of object within the Domain in which it is defined. The unique identification of such an object requires the specification of the name of the Domain and the name of the object, thus implying a two level hierarchy.

#### **7.3.1.3 Application Association-specific Scope**

A name having “Application Association-specific” (AA-specific) scope may only be referenced by the client SMS-user for which the name was defined, and only on the specific application association over which the name’s definition is valid. The definition of an object bearing an application association-specific name, if not explicitly deleted previously, shall be deleted whenever the defining application association ceases to exist.

### **7.3.2. CLASSES OF OBJECTS**

The specific instances of SMS objects which can be named are listed in Table 7-1. Not all objects can have all possible Name Scopes. The allowed combinations are indicated by Xs in the table.

**Table 7-1 - SMS Objects**

	VSD	Domain	AA	Section
Named Variable Objects	X	X	X	12
ScatteredAccess Objects	X	X	X	12
Named Variable List Objects	X	X	X	12
Named Type Objects	X	X	X	12
Semaphore Objects	X	X	X	13
Event Condition Objects	X	X	X	15
Event Action Objects	X	X	X	15
Event Enrollment Objects	X	X	X	15
Journal Objects	X		X	16
Domain Objects	X			10
Program Invocation Objects	X			11

In addition to the requirement that the name of each object be unique within its Name Scope and for the class of the object, there is an additional requirement on the first two objects listed. Named Variable Objects and Scattered Access Objects share a common name space. Therefore the name of any such object shall be unique within this common name space for its Name Scope.

#### NOTE

There are other limitations on the names of Event Conditions, Event Actions, and Event Enrollments. Creation of a semaphore automatically creates an Event Condition of the same name (see 13.4). Therefore, a semaphore cannot be proposed for creation which has the same name as an existing Event Condition. If the monitor parameter is selected, the creation of a Program Invocation automatically creates an Event Condition, and Event Action, and an Event Enrollment, all having the same name as the Program Invocation (see 11.2). Therefore, a Program Invocation cannot be proposed for creation whose name is the same as an existing Event Condition, Event Action, or Event Enrollment.

### 7.3.3. OBJECT LIFETIME

Each of the objects listed above has a lifetime within the VSD which can be inferred from the scope of its name.

Such objects exist as long as the VSD exists (unless explicitly deleted).

#### Domain-specific Scope

Such object exists as long as the Domain on which they depend exists (unless explicitly deleted).

#### AA-specific Scope

Such object exists only as long as the Application Association over which they were defined continues to exist (unless explicitly deleted).

#### **7.3.4. OBJECT VISIBILITY**

The Name Scope of a named object also determines the visibility of the object. An object whose name is VSD-specific or Domain-specific may be referenced by any association to the VSD; an object whose name is AA-specific may only be referenced by the association over which the object was defined.

#### **7.3.5. CREATION OF SMS OBJECTS**

Each of the SMS objects can either be static, that is pre-defined within the implementation, or dynamic, coming into existence during the course of operation of the VSD. Static objects are usually not SMS deletable, while dynamic objects are usually SMS deletable, but there may be exceptions to wither rule. Static objects are pre-defined by the implementation and have names assigned to them either by the implementor, or in conformance to this Standard or to one of the Companion Standards.

Dynamic objects can come into existence either (1) through explicit SMS service procedures, (2) through local action of the system or of the system operators, or (3) through the execution of some Program Invocation. The means of local creation is outside the scope of this Standard. However, if a locally created object is to be visible to SMS services, its external behavior, including all its visible attributes, shall be consistent with this service definition.

#### **7.3.6. DELETION OF SMS OBJECTS**

All SMS objects which have the attribute SMS Deletable equal to TRUE may be removed from the VSD through appropriate SMS service requests. In addition to this explicit method of deletion, SMS objects may also be deleted through local action of the system or of the system operators or through the execution of some Program Invocation. The means of local deletion is outside the scope of this Standard. Objects which are subordinate to a Domain are automatically deleted when the Domain is deleted. This is true even if the subordinate object has its SMS-Deletable attribute set to FALSE.

#### **7.3.7. ALTERATION OF SMS OBJECTS**

All SMS objects have as part of their description a list of externally visible attributes. In addition to the SMS services which alter these attributes, these attributes may also be changed through the local action of the system or of the system operators, or through the execution of a Program Invocation. In particular, the values of variables associated with Domains or with the VSD directly may change due to the execution of a Program Invocation. The means of local alteration of Object Attributes is outside the scope of this Standard.

A VSD should, if possible, guarantee that access to any SMS object required by an SMS service is not interruptible. In other words, for the entire duration of the SMS service, the object should have a set of attributes that represent the state of the VSD at a single instant of time. Since such guarantees may not be possible, or if possible for some object may not be possible for all objects, the static conformance statement for the VSD

shall state whether uninterruptible access is supported and, if supported, under what constraints it is guaranteed.

## 7.4. OBJECT NAME STRUCTURE

The parameter Object Name occurs frequently in the specification of SMS services. The structure of the Object Name is shown in Table 7-2.

**Table 7-2 - Object Name**

Object Name	Req Rsp	Ind Cnf
Name Scope	M	M(=)
AA-specific	S	S(=)
Item Identifier	M	M(=)
Domain-specific	S	S(=)
Domain Identifier	M	M(=)
Item Identifier	M	M(=)
VSD-specific	S	S(=)
Item Identifier	M	M(=)

### 7.4.1. NAME SCOPE

This parameter specifies which of the possible scopes is to be used for this Object Name

### 7.4.2. AA-SPECIFIC

Some named objects may have an AA-specific name. Such names must be defined over the association on which they exist.

#### 7.4.2.1. Item Identifier

This parameter, of type Identifier, is the name of the object. This parameter shall be unique within this application association for the class of object named.

### 7.4.3. DOMAIN-SPECIFIC

Named object which depend on a Domain may have names which are Domain-specific.

#### 7.4.3.1. Domain Identifier

This parameter, of type Identifier, is itself a VSD-specific name of the Domain which contains the named object.

#### 7.4.3.2. Item Identifier

This parameter, of type Identifier, is the name of the object within the named Domain. This parameter shall be unique within the named Domain for the class of object named.

#### **7.4.4. VSD-SPECIFIC**

Any named object may have a VSD-specific name.

##### **7.4.4.1. Item Identifier**

This parameter, of type Identifier, is the name of the object. This parameter shall be unique within the VSD for the class of object named.

#### **7.5. SERVICES ON THE VSD**

Section 9 of this Standard describes the SMS services which operate on the VSD in its entirety. In addition to these services, provision is made for groups defining Companion Standards to define additional services which operate on the VSD.

## **8. GENERAL MANAGEMENT SERVICES**

[Note: Major deviations of SMS from the MMS standard are addressed in the general Management Services. To accommodate space communications and long delays, the MMS parameters that require negotiation will be retained but pre-defined. That is, rather than requiring a negotiation for each individual device or spacecraft, the negotiated parameters can be known to both sides. The `initiate.request` that intentionally uses the pre-defined values and will not require an `initiate.confirm` before transitioning to the SMS Environment. In addition, an `reboot.request` is defined to support emergency conditions. The MMS standard did not make provisions for such a service. In the MMS view, a factory device that would not response to an `initiate.request` would require (for safety reasons) a visit from supervisory personal. This is most often not possible in the space environment.]

### **8.1.1. INTRODUCTION**

The environment and general management services contain the Initiate, Conclude, Abort, Cancel and Reject services. These services allow the SMS-user:

- a) to initiate communication with another SMS-user in the SMS environment, and to establish the requirements and capabilities that support that communications;
- b) to conclude communication with another SMS-user in the SMS environment in a graceful manner;
- c) to abort communications with another SMS-user in the SMS environment in an abrupt manner;
- d) to cancel pending service requests;
- e) to receive notification of protocol errors that occur.

### **8.1.2. ENVIRONMENT MANAGEMENT STATE DIAGRAM**

This section defines the state diagram for entering and leaving the SMS environment. The initial state for both the calling and called SMS users shall be the state of “NO SMS Environment”. The state diagram is depicted from the point of view of an SMS-user.

#### **8.1.2.1. The “No SMS Environment” State**

While in the state “No SMS Environment”, an SMS-user shall only issue the `initiate.request` service primitive. As a local matter, the SMS-user can issue the `initiate.request` and transition through the Establishing SMS Environment to the SMS-Environment without receiving the `initiate.confirm` if the negotiated parameters are pre-defined and not a matter for negotiation.

[ Note: MMS is a connection oriented protocol that requires the establishment of a connect before any other messages can be exchanged. While this mode of operation can be supported for onboard communications and for communications

where response delays are less than 250 seconds, it would be prohibitive for spacecraft operating in deep space with round trip light times of many minutes or hours. In addition, the connection oriented mode does not provide for an emergency mode of operation required when a spacecraft experiences serious problems. This provision is intended to address the problems encountered with long round trip delays experienced in communication with deep space probes.]

### **8.1.2.2. The “Establishing SMS Environment “ (Calling) State**

While in the “Establishing SMS Environment (Calling) “, an SMS -user shall only issue an abort.request service primitive.

### **8.1.2.3. The “Establishing SMS Environment “ (Called) State**

While in the “Establishing SMS Environment (Called) “, an SMS -user shall only issue the initiate.response or abort.request service primitive.

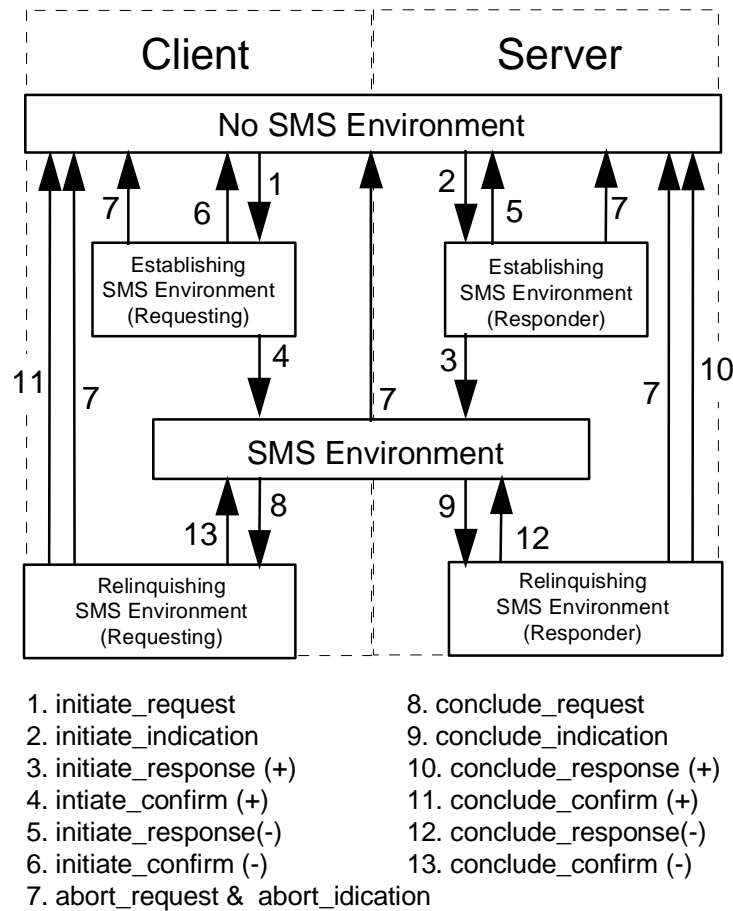
### **8.1.2.4. The “SMS Environment “ State**

Sections 8 to 16 place requirements on the use of service primitives in the state “SMS Environment”. This state is divided into a series of sub-states, which are described in those sections. All SMS states except the “No SMS Environment”, “Establishing SMS Environment (Calling)”, “Establishing SMS Environment (Called)”, “Relinquishing SMS Environment (Requester)” and Relinquishing SMS Environment (Responder)” are sub-states of the “SMS Environment “ state.

The only events which cause an exit from the “SMS Environment” state are the issuance of an abort.request, the issuance of a conclude.request service primitive, the receipt of an abort.indication, or the receipt of a conclude.indication service primitive.

While in the state “SMS Environment”, an SMS-user may issue any request or response service primitive, subject to the following restrictions:

- a) other sections of this Standard restrict the usage of services and service primitives via sequencing requirements;
- b) a response primitive shall not be issued unless a request primitive for a corresponding service indication has been received (see section 7 of part 2 of this Standard);
- c) the initiate.request service primitive shall not issued.



**Figure 8-1 - The state diagram for entering and leaving the SMS environment**

[ Note: In the state diagram above, Figure 8-1, it is possible (though not reflected yet) for an SMS-user may transition from the “Establishing SMS Environment (Calling)” state to the “SMS Environment” without receipt of an initiate.confirm(+) {4}. Support for this capability requires that all negotiated parameters be pre-defined and set at the time the initiate.request is issued and that the pre-defined negotiated parameters are in fact the parameters required by the SMS-provider . ]

#### **8.1.2.5. The “Relinquishing SMS Environment (Requester)” State.**

While in the state “Relinquishing SMS Environment (Requester)”, the only request primitive that the SMS-user may issue is the abort.request service primitive. Note that responses, either (+) or (-) may continue to be issued.

#### **8.1.2.6. The “Relinquishing SMS Environment (Responder)” State.**

While in the state “Relinquishing SMS Environment (Responder)”, the SMS-user may only issue the abort.request and conclude.response service primitives.

## **8.2. INITIATE SERVICE**

The Initiate service shall be used to establish the SMS environment and to allow the communicating SMS-users to exchange information regarding their capabilities and requirements. Initiate service can be employed as a confirmed or unconfirmed service.

The confirmed Initiate service must complete successfully before any other services may be carried out between a pair of SMS-users in the SMS environment. The confirmed Initiate service is to be used in situations where there is a minimum of response delay. The confirmed Initiate service would be used between onboard systems, between spacecraft, and between spacecraft and ground systems.

The unconfirmed Initiate service establishes the SMS environment under conditions where the delay between the send and receive preclude a timely response. The unconfirmed service is intended for use between spacecraft, between spacecraft and ground systems.

The Initiate service shall be used to establish the MMS environment and to allow the communicating MMS-users to exchange information regarding their capabilities and requirements. The Initiate service must complete successfully before any other services may be carried out between a pair of MMS-users in the MMS environment.

[Note: the MMS Initiate service is a confirmed service requiring the server to respond positively before any other MMS service can be exchanged. While appropriate in a factory environment, this requirement would be impossible to accommodate in deep space mission environment with a 8 hour round-trip communications path. ]

### 8.2.1. STRUCTURE

The structure of the component service primitives is shown in Table 8-1.

**Table 8-1 - Structure of Component Service Primitives**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Argument (COMP, SMS)	M	M			
Local Detail Calling	U	U(=)			
Proposed Max Serv Outstanding Calling	M	M			
Proposed Max Serv Outstanding Called	M	M			
Proposed Data Structure Nesting Level	U	U			
Init Request Detail	M	M			
Result (+) (COMP, SMS)			S	S(=)	
Local Detail Calling			U	U(=)	
Negotiated Max Serv Outstanding Calling			M	M(=	
Negotiated Max Serv Outstanding Called			M	)	
Negotiated Data Structure Nesting Level			U	M(=	
Init Response Detail			M	)	
Result (-)			S	U(=)	
Error Type			M	M(=	
				)	
				S(=)	
				M(=	
				)	

#### 8.2.1.1. Argument

This parameter shall convey the service specific parameters of the Initiate service request.

##### 8.2.1.1.1. Local Detail Calling

When present, this parameter, of type integer, shall represent information about the Calling MMS-User's implementation. The content and interpretation of this field is a local matter and not subject for further standardization.

##### 8.2.1.1.2. Proposed Max Serv Outstanding Calling

This parameter, of type integer, shall identify the proposed maximum number of Transaction Object instances whose Application Association Identifier attribute references this association that may be created at the Calling MMS-user.

The value of this parameter may be reduced by the MMS-provider. The value is the indication primitive shall be less than or equal to the value in the request primitive. The value in the request or indication primitives shall not be less than zero.

### 8.2.1.1.3. Proposed Max Serv Outstanding Called

This parameter, of type integer, shall identify the proposed maximum number of Transaction Object instances that may be created at the Called MMS-user whose Application Association Identifier attribute references this association.

The value of this parameter may be reduced by the MMS-provider. The value is the indication primitive shall be less than or equal to the value in the request primitive. The value in the request or indication primitives shall not be less than zero.

### 8.2.1.1.4. Proposed Data Structure Nesting Level

This parameter, of type integer, shall indicate the proposed maximum number of levels of nesting supported by both of the MMS-users that may occur within any data element used in the association. Absences of this parameter shall indicate unlimited number of nesting levels.

The request and indication primitives shall specify the maximum nesting level of Type Specification (explicit or derived) that the calling MMS-user desires to use in the negotiated MMS environment. A value of zero shall indicate that only simple types are allowed.

The value of this parameter may be reduced by the MMS-provider. The value is the indication primitive shall be less than or equal to the value in the request primitive.

### 8.2.1.1.5. Init Request Detail

This parameter shall specify additional information necessary for the establishment of an instance of MMS communications. For the abstract syntax defined in part 1 and 2 of this Standard, this parameter is described in section 8.2.3.

### **8.2.1.2. Result (+)**

This parameter shall indicate that the requested service has succeeded. When success is indicated, the following additional parameters shall also apply.

#### 8.2.1.2.1. Local Detail Calling

When present, this parameter, of type integer, shall represent information about the Called MMS-User's implementation. the content of this field is a local matter and not a subject for further standardization,

#### 8.2.1.2.2. Negotiated Max Serv Outstanding Calling

When present, this parameter, of type integer, shall identify the maximum number of Transaction Object instances that may be created at the calling MMS-user whose Application Association Identifier attribute references this association.

## SuperMOCA Space Messaging Service

The negotiated maximum number of services outstanding in the response primitive shall be less than or equal to the Proposed Max Serv Outstanding Calling parameter in the indication primitive, but shall not be less than zero.

### 8.2.1.2.3.Negotiated Max Serv Outstanding Called

When present, this parameter, of type integer, shall identify the maximum number of Transaction Object instances that may be created at the called MMS-user whose Application Association Identifier attribute references this association.

The negotiated maximum number of services outstanding in the response primitive shall be less than or equal to the Proposed Max Serv Outstanding Called parameter in the indication primitive, but shall not be less than zero.

### 8.2.1.2.4.Negotiated Data Structure Nesting Level

This parameter, of type integer, shall indicate the proposed maximum number of levels of nesting supported by both of the MMS-users that may occur within any data element used in the association. Absences of this parameter shall indicate unlimited number of nesting levels.

This parameter shall be equal to or less than the Proposed Data Structure Nesting level parameter in the indication primitive, but shall not be less than zero.

The response and confirmation primitives shall specify the maximum nesting level of Type Specification (explicit or derived) that shall be used in the negotiated MMS environment. A value of zero shall indicate that only simple types are allowed.

### 8.2.1.2.5.Init Response Detail

This parameter shall specify additional information necessary for the establishment of an instance of MMS communications. For the abstract syntax defined in part 1 and 2 of this Standard, this parameter is described in section 8.2.4.

### **8.2.1.3. Result (-)**

This parameter shall indicate that the service request failed. The Error Type parameter, which is defined in detail in section 17, provides the reason for failure.

## **8.2.2. SERVICE PROCEDURE**

The called MMS-user shall issue a response primitive indicating success in the Result parameter if that MMS-user is willing to accept communications in the MMS environment under the constraints identified in the indication primitive, or if alternate values can be proposed (according to the negotiation rules), with the requesting MMS-user. Otherwise, a response primitive indicating the Result(-) parameter shall be issued.

Successful execution to the Initiate service shall result in the establishment of the MMS environment. The MMS environment shall only be established through the use of the Initiate service. The Initiate service shall not be used within an established MMS

environment. If an Initiate Request PDU is received on an association where an established MMS environment exists, a Reject PDU shall be issued with a Reject PDU Type PDU-ERROR and a Reject Code of ILLEGAL-ACSE-MAPPING.

### 8.2.3. INIT REQUEST DETAIL PARAMETER

The structure of the Init Request Detail Parameter is shown in Table 8-2.

**Table 8-2 - Structure of the Init Request Detail Parameter**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Proposed Version Number	M	M			
Proposed Parameter CBB	M	M			
Services Supported Calling	M	M			

#### 8.2.3.1. Proposed Version Number

This parameter, of type integer, shall contain a number which represents a minor version number of parts 1 and 2 of this standard. The minor version number of parts 1 and 2 of this Standard shall be specified in section 17 of part 2 of this Standard. This parameter, the Proposed Version Number, is the proposed minor version number which will be used in the presentation Standard for this instance of communication. Proposal of a number greater than one indicates support, in this presentation context, for all minor versions between one and the number proposed, inclusive. The value of this parameter may be reduced by the MMS-provider if it cannot support the requested value. The value in the indication primitive shall be less than or equal to the value in the request primitive, but not less than one.

#### 8.2.3.2. Proposed Parameter CBB

This parameter, of type bitstring, shall specify the set of parameters conformance building blocks (CBB) which are proposed to be supported in the presentation context derived from the abstract syntax defined section 19 of part 2 of this standard on this application association.

The value of this parameter in the request primitive shall specify the set of Parameter CBBs supported by the Calling MMS-user.

The value of this parameter in the indication primitive shall specify the intersection of the set of Parameter CBBs supported by the Calling MMS-user and the set of Parameter CBBs supported by the MMS-provider.

The Possible parameter conformance building blocks are specified in section 18 of part 2 of this Standard. The assignment of a parameter CBB to an individual bit of a CBB bitstring type is specified in section 8 of part 2 of this Standard. A value of one in the assigned bit shall indicate support for this corresponding CBB. A value of zero shall

indicate non-support. All bits shall be encoded and any additional bits received shall be ignored.

### 8.2.3.3. Services Supported Calling

This parameter, of type bitstring, shall specify support by the Calling MMS-user of the set of services for use in the presentation context derived from the abstract syntax defined in section 19 of part 2 of this Standard on the application association.

The value of the parameter in the indication primitive shall specify the intersection of the set of services supported by the Calling MMS-user and the set of services supported by the MMS-provider.

The assignment of an service to an individual bit of the bitstring type is specified in section 19 of part 2 of this Standard. A value of one in the assigned bit shall indicated support for the corresponding service. A value of zero shall indicate non-support. All bits shall be encoded and any additional bits received shall be ignored.

Support for confirmed services shall be defined as the ability to receive a request indication and execute the service procedure defined as the responder role. Support of unconfirmed services shall be defined as the ability to accept and indication primitive and to pass the parameters to the service interface. Support of a modifier shall be defined as the ability to accept an indication primitive that contains the Modifier and to execute properly the service procedure defined for the Modifier.

If a confirmed service, an unconfirmed service, or Modifier is supported, then a Reject PDU shall not be issued on receipt of that service or modifier service, except in the case of a protocol error. If a confirmed service, an unconfirmed service, or modifier is not supported, then a Reject PDU shall be issued on receipt of that service or modified service with a reject code of “UNRECOGNIZED SERVICE”.

### 8.2.4. INIT RESPONSE DETAIL PARAMETER

The structure of the Init Response Detail Parameter is shown in Table 8-3.

**Table 8-3 - Structure of the Init Response Detail Parameter**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Negotiated Version Number			M	M(=)	
Negotiated Parameter CBB			M	M(=)	
Services Supported Called			M	M	

#### 8.2.4.1. Negotiated Version Number

This parameter, of type integer, shall contain a number which represents a minor version number of parts 1 and 2 of this Standard. The minor version number of parts 1 and 2 of this Standard shall be that specified in section 17 of part 2 of this Standard. This parameter, the Negotiated Version Number, shall be the minor version number which will be used in the presentation context derived from the abstract syntax defined in section 19

of part 2 of this Standard for this instance of communication. The number shall be less than or equal to the Proposed Version Number parameter in the request primitive. It shall not be reduced to less than one.

#### **8.2.4.2. Negotiated Parameter CBB**

This parameter, of type bitstring, shall specify the negotiated set of parameter conformance building blocks (CBB) which are to be supported in the presentation context derived from the abstract syntax defined in section 19 of part 2 of this Standard on this application association.

The value of this parameter in the response primitive shall specify the intersection of the set of Parameter CBBs supported by the Called SMS-user and the set of Parameter CBBs specified by the Proposed Parameter CBB parameter in the indication primitive.

The possible parameter conformance building blocks are specified in section 18 of part 2 of this Standard. The assignment of a Parameter CBB to an individual bit of a CBB bitstring type is specified in section 8 of part 2 of this Standard. A value of one in the assigned bit shall indicate support for the corresponding CBB. A value of zero shall indicate non-support. All bits shall be encoded and any additional bits received shall be ignored.

#### **8.2.4.3. Services Supported Called**

This parameter, of type bitstring, shall specify support by the Called SMS-user of a set of services for use in the presentation context derived from the abstract syntax defined in this Standard on the application association.

The value of the parameter in the confirmation primitive shall specify the intersection of the set of services supported by the Called SMS-user and the set of services supported by the SMS-provider.

The assignment of a service to an individual bit of the bitstring type is specified in section 8 of part 2 of this Standard. A value of one in the assigned bit shall indicated support for the corresponding service. A value of zero in the assigned bit shall indicated non-support. All bits shall be encoded and any additional bits received shall be ignored.

Support for confirmed services shall be defined as the ability to receive an indication primitive and properly execute the service procedure defined for the responder role. Support of unconfirmed services shall be defined as the ability to accept an indication primitive and to pass the parameters to the service interface. Support of a modifier shall be defined as the ability to accept an indication primitive that contains the Modifier and to properly execute the service procedure defined for the Modifier.

If a confirmed service, an unconfirmed service, or Modifier is supported, then a Reject PDU shall not be issued on receipt of that service of modified service, except in the case of a protocol error. If a confirmed service, an unconfirmed service, or Modifier is not supported, then a Reject PDU shall be issued on receipt of that service of modified service with a reject code of "UNRECOGNIZED SERVICE".

### 8.3. CONCLUDE SERVICE

The Conclude service may be used to cause the orderly relinquishing of the SMS environment. An SMS-user requests the Conclude service to indicate that it has completed requests which it had planned and that no further requests will be issued.

#### 8.3.1. STRUCTURE

The structure of the component service primitive is shown in Table 8-4.

**Table 8-4 - Conclude Service**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Argument	M	M(=)			
Result(+)			S	S(=)	
Result(-) Error Type			S M	S(=) M(=)	

##### 8.3.1.1. Argument

There are no service specific parameters for Conclude service request.

##### 8.3.1.2. Result(+)

This parameter shall indicate that the requested service has succeeded.

##### 8.3.1.3. Result(-)

This parameter shall indicate that the service request failed. The Error Type parameter, which is defined in detail in section 17, provides the reason for failure.

### 8.3.2. SERVICE PROCEDURE

The requesting SMS-user shall issue a Conclude request primitive, which begins the Conclude service. Once this primitive has been issued, the requesting SMS-user shall not issue any further request primitives until a Conclude confirm primitive is received from the SMS-provider (except the Abort request primitive, which may be issued at any time). The requesting SMS-user may continue to issue response primitives in order to complete service requests from the peer SMS-user.

The requesting SMS-user shall not issue a Conclude request primitive to a peer on an association if a Domain with a State attribute value equal to LOADING, COMPLETE, INCOMPLETE, D1, D2, D3, or D9 exists at the requesting SMS-user as a result of a request from the peer SMS-user on the association.

## SuperMOCA Space Messaging Service

In the peer open system, the responding SMS-user shall issue a Conclude response primitive indicating whether or not it accepts the conclusion (via the Result parameter) before any other service request primitives may be issued by that SMS-user (except the Abort request service primitive, which may be issued at any time). The SMS-user shall not accept conclusion on an association if it is awaiting any response from the peer SMS-user for a confirmed service or the Conclude Service.

A responding SMS-user shall not accept a Conclude attempt on a association if any of the following conditions exists:

- a) The responding SMS-user has received an indication on the association for a confirmed service request from the peer SMS-user for which it has not issued a response;
- b) An Upload State Machine exists on the responding SMS-user as a result of a request from the peer SMS-user on the association;
- c) A Domain with a State attribute value equal to LOADING, COMPLETE, or INCOMPLETE exists on the responding SMS-user as a result of a request from the peer SMS-user on the association;
- d) The responding SMS-user has control of a semaphore on the requesting SMS-user on the association;
- e) The requesting SMS-user has control of a semaphore on the responding SMS-user on the association.

Upon issuing a Conclude response primitive with a Result(+) parameter, the SMS Environment shall be considered terminated. No further request or response primitives shall be issued by the responding SMS-user in that SMS environment on that association.

Upon issuing a Conclude response primitive with a Result(-) parameter, the SMS Environment shall not be affected and the responding SMS-user may continue to issue request and/or response service primitives.

Upon receiving a Conclude confirm primitive with a Result(+), the requesting SMS-user shall consider the SMS environment to be terminated, and no further request or response primitives shall be issued by the requesting SMS-user in the SMS environment on that association.

Upon receiving a Conclude confirm primitive with a Result(-), the requesting SMS-user shall consider the SMS Environment to be unaffected, and may continue to issue request or response service primitives in the SMS environment.

### NOTE 1

The effect of failed Conclude service confirmation is as if no Conclude request had ever been issued.

## SuperMOCA Space Messaging Service

Upon successful completion of the Conclude service, all application association specific objects associated with the terminated MMS environment are released unless other specifications for that object are explicitly stated in this Standard.

### NOTE 2

It is possible that both of the peers will initiate a Conclude attempt at or near the same time resulting in failure of both attempts. If both peers retry the Conclude, on result of failure, then this situation could potentially (although very unlikely) continue indefinitely. Through use of application design, this situation can be avoided. One possible method is to designate the Calling SMS-user as the peer that will initiate the Conclude attempt should this situation arise.

## 8.4. ABORT SERVICE

The Abort service shall be used to relinquish the SMS environment abruptly and without negotiation. The SMS-user shall issue the Abort request primitive to indicate that it wishes immediately, and without negotiation, to discontinue communications in the SMS Environment on the application association. The effect of the Abort service may be to destroy previously issued requests and/or responses issued by either of the SMS-users. The SMS Abort service is derived from and makes use of the A-ABORT service of ACSE (see 17.2 in part 2 of this Standard).

### NOTE

The abort indication service primitive may also be generated by the SMS-provider.

### 8.4.1. STRUCTURE

The structure of the component service primitive is shown in Table 8-5.

**Table 8-5 - Abort Service**

Parameter Name	Req	Ind	CBB
Argument Locally Generated	M	M M	

#### 8.4.1.1. Argument

This parameter shall convey the service specific parameters of the Abort service. The abort service primitives additionally shall contain the parameters specified for the ACSE abort service to which the SMS Abort is mapped. These parameters are specified in ISO 8649.

##### 8.4.1.1.1. Locally Generated

This parameter, of type boolean, shall indicated whether the abort request was generated by the system in which the SMS-user receiving the indication is located

(indicated by true value), or whether the abort request was received by that system (indicated by the false value). This parameter shall be provided by the SMS-provider.

### **8.4.2. SERVICE PROCEDURE**

In the case of an Abort initiated by the SMS-user, the peer SMS-user shall be notified of the user requested abort by receipt of an Abort indication service primitive, and the SMS Environment shall be terminated.

In the case of an Abort initiated by the SMS-provider, both SMS-users shall be notified of the provider abort by receipt of an Abort indication primitive (if this is possible), and the SMS Environment shall be terminated.

Upon completion of the Abort service, all application association specific objects associated with the terminated SMS environment are released unless other specifications for that object are explicitly stated in this Standard.

Upon termination of the SMS Environment, all application association specific objects associated with the terminated SMS environment are released unless other specifications for that object are explicitly stated in this Standard.

### **8.5. CANCEL SERVICE**

The Cancel service is used by an SMS-user to cancel a request that has previously been issued, but has not yet completed. Only confirmed services may be canceled (see section 5).

**8.5.1. STRUCTURE**

The structure of the component service primitive is shown in Table 8-6.

**Table 8-6 - Cancel Service**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Argument	M	M(=)			
Original Invoke ID	M	M(=)			
Result(+)			S	S(=)	
Original Invoke ID			M	M(=)	
Result(-)			S	S(=)	
Original Invoke ID			M	M(=)	
Error Type			M	M(=)	

**8.5.1.1. Argument**

This parameter shall convey the service parameters for the Cancel service request.

**8.5.1.1.1.8.5.1.1.1 Original Invoke ID**

This parameter, of type integer, shall specify the invoke ID of the service whose cancellation is desired. (An invoke ID is provided for every confirmed service request primitive, see section 5.)

**8.5.1.2. Result(+)**

The Result(+) parameter shall indicate that the Cancel service request succeeded. When success is indicated service specific parameters shall also be returned.

**8.5.1.2.1.8.5.1.2.1 Original Invoke ID**

This parameter, of type integer, shall specify the invoke ID of the service that was canceled. (An invoke ID is provided for every confirmed service request primitive. See section 5.)

**8.5.1.3. Result(-)**

The Result(-) parameter shall indicate that the Cancel service request failed. The Error Type parameter, which is defined in detail in section 17, provides the reason for failure.

**8.5.1.3.1.Original Invoke ID**

This parameter, of type integer, shall specify the invoke ID of the service whose cancellation was desired. (An invoke ID is provided for every confirmed service request primitive. See section 5.)

#### 8.5.1.3.2. Error Type

The Error Type parameter, which is defined in detail section 17, provides the reason for the failure.

### 8.5.2. SERVICE PROCEDURE

Upon receiving the Cancel indication service primitive, the responding SMS-user shall attempt to cancel the designated service, assuming that it can do this non-destructively and can return to a state that is logically as if the service indication targeted for cancellation had not been received. Destructive cancellation is permitted for the Start, Stop, Resume, and Reset Services provided it follows the service procedures defined for these services. If this is not the case, or if the service request invocation identified has not been requested, or if a response primitive has already been issued, or if the cancelable attribute of the Transaction object associated with the designated services has a value of false, then the Cancel request shall fail. In this case, the responding SMS-user shall issue a Cancel response service primitive with the Result(-) parameter with the appropriate Cancel error codes for the Error Type parameter. The responding SMS-user shall continue with the original service and eventually return a response, either positive or negative, as appropriate.

Otherwise, the service invocation identified shall be successfully canceled by the responding SMS-user, and a Cancel response primitive with the Result(+) parameter shall be issued. The responding SMS-user shall also issue a response primitive with a Result(-) parameter for the canceled service indicating the responder shall then appear logically as is the service indicating the SERVICE-PREEMPT Error Class with an Error Code of CANCEL. The state of the responder shall then appear logically as if the service targeted for cancellation had not been requested, unless otherwise stated in the service procedure for the canceled service. Addition limitations on requesting the cancel service are specified in section 6 of part 2 of this Standard.

### 8.6. REJECT SERVICE

The Reject service is a provided-initiated service that is used to inform the SMS-user of the occurrence of a protocol error. A definition of a protocol error may be found in section 3 of part 2 of this Standard.

#### 8.6.1. STRUCTURE

The structure of the component service primitive is shown in Table 8-7.

**Table 8-7 - Reject Service**

Parameter Name	Ind	CBB
Detected Here	M	
Original Invoke ID	C	
Reject PDU Type	M	

Reject Code	M	
-------------	---	--

#### 8.6.1.1. Detected Here

This parameter, of type boolean, shall indicate whether the protocol error that results in the Reject service was detected at the local end or remote end of the underlying presentation connection. If true, the protocol error was detected at the local end, otherwise the error was detected at the remote end.

#### NOTE

Handling of local error conditions between an SMS-user and SMS-provider is a local matter.

#### 8.6.1.2. Original Invoke ID

This parameter, of type integer, shall indicate the original invoke ID of the PDU which was found to be in error. Its presence is conditional on whether the invoke ID could be determined. If there is no invoke ID specified in the PDU being rejected, this parameter shall not be present.

#### 8.6.1.3. Reject PDU Type

This parameter, of type integer, shall indicate the type of the PDU that caused the protocol error. The value PDU-ERROR shall be used when the PDU being rejected is not a syntactically valid SMS PDU. The possible values are as follows:

CONFIRMED-REQUESTPDU  
 CONFIRMED-RESPONSEPDU  
 CONFIRMED-ERRORPDU  
 UNCONFIRMEDPDU  
 PDU-ERROR  
 CANCEL-REQUESTPDU  
 CANCEL-RESPONSEPDU  
 CANCEL-ERRORPDU  
 CONCLUDE-REQUESTPDU  
 CONCLUDE-RESPONSEPDU  
 CONCLUDE-ERRORPDU

#### 8.6.1.4. Reject Code

*8.6.1.4.1.1. This parameter, of type integer, shall indicate additional information regarding the reason for the Reject within a given Reject PDU Type parameter value. The possible values for this parameter are given below under each possible Reject PDU Type parameter value.*

8.6.1.4.2. Codes For CONFIRMED-REQUESTPDU

8.6.1.4.2.1.1. OTHER

## SuperMOCA Space Messaging Service

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

### 8.6.1.4.3.UNRECOGNIZED-SERVICE

8.6.1.4.4.This code shall be used when the service requested is not supported or is not recognized.

#### 8.6.1.4.4.1.UNRECOGNIZED-MODIFIER

This code shall be used when the modifier requested is not supported or is not recognized.

#### 8.6.1.4.4.2.UNRECOGNIZED-INVOKEID

This code shall be used when an invoke ID does not meet the requirements of this Standard.

#### 8.6.1.4.4.3.INVALID-ARGUMENT

This code shall be used when the service argument does not meet the requirements of this Standard.

#### 8.6.1.4.4.4.INVALID-MODIFIER

This code shall be used when the service modifier does not meet the requirements of this Standard.

#### 8.6.1.4.4.5.MAX-SERV-OUTSTANDING-EXCEEDED

This code shall be used when the negotiated maximum number of confirmed services that may be outstanding is exceeded by a confirmed service request that is received.

#### 8.6.1.4.4.6.MAX-RECURSION-EXCEEDED

This code shall be used when the PDU received exceeds the negotiated maximum data structure nesting level.

#### 8.6.1.4.4.7.VALUE-OUT-OF-RANGE

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

### 8.6.1.4.5.Codes For CONFIRMED-RESPONSEPDU

#### 8.6.1.4.5.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### 8.6.1.4.5.2.UNRECOGNIZED-SERVICE

## SuperMOCA Space Messaging Service

This code shall be used when the service specified is not supported, is not recognized, or is not the same service that was requested with the invoke ID specified in the PDU.

### *8.6.1.4.5.3.INVALID-INVOKEID*

This code shall be used when an invoke ID does not meet the requirements of this Standard, or no confirmed service has been requested for the specified invoke ID.

### *8.6.1.4.5.4.INVALID-RESULT*

This code shall be used when the service result does not meet the requirements of this Standard.

### *8.6.1.4.5.5.MAX-SERV-OUTSTANDING-EXCEEDED*

This code shall be used when the PDU received exceeds the negotiated maximum data structure nesting level.

### *8.6.1.4.5.6.VALUE-OUT-OF-RANGE*

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

### *8.6.1.4.6.Codes For CONFIRMED-ERRORPDU*

#### *8.6.1.4.6.1.OTHER*

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### *8.6.1.4.6.2.UNRECOGNIZED-SERVICE*

This code shall be used when the service specified is not supported, is not recognized, or is not the same service that was requested with the invoke ID specified in the PDU.

#### *8.6.1.4.6.3. INVALID-INVOKEDID*

This code shall be used when an invoked ID does not meet the requirements of this Standard, or no confirmed service has been requested for the specified invoked ID.

#### *8.6.1.4.6.4.INVALID-SERVICEERROR*

This code shall be used when the service error does not meet the requirements of this Standard.

#### *8.6.1.4.6.5.VALUE-OUT-OF-RANGE*

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

## SuperMOCA Space Messaging Service

### 8.6.1.4.7.Codes For UNCONFIRMEDPDU

#### *8.6.1.4.7.1.OTHER*

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### *8.6.1.4.7.2.UNRECOGNIZED-SERVICE*

This code shall be used when the service specified is not supported or is not recognized.

#### *8.6.1.4.7.3.INVALID-ARGUMENT*

This code shall be used when the service argument does not meet the requirements of this Standard.

#### *8.6.1.4.7.4.MAX-RECURSION-EXCEEDED*

This code shall be used when the PDU received exceeds the negotiated maximum data structure nesting level.

#### *8.6.1.4.7.5.VALUE-OUT-OF-RANGE*

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

### 8.6.1.4.8.Codes For PDU-ERROR

#### *8.6.1.4.8.1.UNKNOWN-PDU-TYPE*

This code shall be used when the PDU type received is not recognized or is not supported.

#### *8.6.1.4.8.2.INVALID-PDU*

This code shall be used when the PDU type received is syntactically incorrect, and further diagnostics cannot be provided due to the severity of the error.

#### *8.6.1.4.8.3.ILLEGAL-ACSE-MAPPING*

This code shall be used when the PDU type received is not properly mapped to an ACSE service primitive.

### 8.6.1.4.9.Codes For CANCEL-REQUESTPDU

#### *8.6.1.4.9.1.OTHER*

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### *8.6.1.4.9.2.INVALID-INVOKEDID*

## SuperMOCA Space Messaging Service

This code shall be used when an invoked ID does not meet the requirements of this Standard.

### 8.6.1.4.10.Codes For CANCEL-RESPONSEPDU

#### 8.6.1.4.10.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### 8.6.1.4.10.2.INVALID-INVOKEDID

This code shall be used when an invoked ID does not meet the requirements of this Standard, or no Cancel service has been requested for the specified invoked ID.

### 8.6.1.4.11.Codes For CANCEL-ERRORPDU

#### 8.6.1.4.11.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### 8.6.1.4.11.2.INVALID-INVOKEDID

This code shall be used when an invoked ID does not meet the requirements of this Standard, or no Cancel service has been requested for the specified invoked ID.

#### 8.6.1.4.11.3.INVALID-SERVICEERROR

This code shall be used when an service error does not meet the requirements of this Standard.

#### 8.6.1.4.11.4.VALUE-OUT-OF-RANGE

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

### 8.6.1.4.12.Codes For CONCLUDE-REQUESTPDU

#### 8.6.1.4.12.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### 8.6.1.4.12.2.INVALID-ARGUMENT

This code shall be used when the argument for the request does not meet the requirements of this Standard.

### 8.6.1.4.13.Codes For CONCLUDE-RESPONSEPDU

#### 8.6.1.4.13.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

#### 8.6.1.4.13.2.INVALID-RESULT

This code shall be used when the service result does not meet the requirements of this Standard.

#### 8.6.1.4.14.Codes For CONCLUDE-ERRORPDU

##### 8.6.1.4.14.1.OTHER

This code shall be used for errors other than those identified in this Standard for this Reject PDU type.

##### 8.6.1.4.14.2.INVALID-SERVICEERROR

This code shall be used when an service error does not meet the requirements of this Standard.

##### 8.6.1.4.14.3.VALUE-OUT-OF-RANGE

This code shall be used when the PDU received contains one or more parameters whose values exceed the range allowed for those parameters by this Standard.

### 8.6.2. SERVICE PROCEDURE

If an SMS-provider receives a RejectPDU it shall issue a Reject indication to the SMS-user. The SMS-user may, as a local matter, use the Abort service to abruptly terminate the SM-environment.

An SMS provider shall be capable of issuing a RejectPDU if it receives a PDU that constitutes a protocol error.

### 8.7. REBOOT SERVICE

The Reboot service is used by an SMS-user to initiate a restart of the executive.

#### 8.7.1. STRUCTURE

The structure of the component service primitive is shown in Table 8-8.

**Table 8-8 - Reboot Service**

Parameter Name	Req	Ind	Rsp	Cnf	CBB
Security Code	M	M(=)			
Result(+)			M	M(=)	
Result(-)			M	M(=)	

#### **8.7.1.1. Security Code**

This parameter, of type octetstring, shall specify an authentication identifier.

#### **8.7.1.2. Result(+)**

The Result(+) parameter shall indicate that the Reboot service request succeeded. When success is indicated service specific parameters shall also be returned.

#### **8.7.1.3. Result(-)**

The Result(-) parameter shall indicate that the Cancel service request failed. The Error Type parameter, which is defined in detail in section 17, provides the reason for failure.

### **8.7.2. SERVICE PROCEDURE**

Upon receiving the Reboot indication service primitive, the responding SMS-user shall authenticate the security and issue a reboot.response primitive with the appropriate indication. If the security code is authenticated, the responding SMS-user will issue a primitive with the Result(+) and reboot the executive. If the security code is not authenticated, the responding SMS-user will issue a primitive with the Result(-) and continue operating.

## **9. VIRTUAL DEVICE SUPPORT SERVICES**

[Note: There are a number of basic services that may be expected from self-identifying devices. For example, a device might be expected to identify its manufacturer, model type, revision number, serial number, and so on. Another example might be the ability of the device to report its status or capabilities. This section defines basic services to support the virtual device. Spacecraft devices would incorporate support for these services to enhance the operability and easy of integration. ]

### **9.1. INTRODUCTION**

The VSD support services contain the Status, UnsolicitedStatus, GetNameList, Identify, Rename and GetCapabilityList services. The services allow the SMS-user to do the following:

- a) get the status of a VSD;
- b) receive an unsolicited message about the status of the VMD;
- c) get lists of various defined objects;
- d) identify the vendor specific attributes of the SMS application entity at the peer system;
- e) change the name of an object;
- f) get lists of the VSD's capabilities.

### **9.2. STATUS SERVICE**

### **9.3. UNSOLICITED STATUS SERVICE**

### **9.4. GETNAMELIST SERVICE**

### **9.5. IDENTIFY SERVICE**

### **9.6. RENAME SERVICE**

### **9.7. GETCAPABILITYLIST SERVICE**

## **10. DOMAIN MANAGEMENT SERVICES**

[Note: Spacecraft devices are likely to incorporate tables and sets of variable as a means to manage their operation. In addition, many spacecraft control systems have been designed to be reprogrammed while in flight. Many manufacturing systems incorporate similar designs. The Domain Management Service provide set of services to define and manage tables, sets of variables and even addressable memory as resources for remote manipulation. A spacecraft device might employ Domain services to manage these assets. However, there is a critical design issue that is exposed by the application of Domain services in the space environment. Specifically, MMS was designed with the premise that a virtual device is always in control of its own resources. Consequently, when a client initiates a download of a domain to a server, the server controls the download process. For example, if a spacecraft executive initiated the download of a program instruction set to a device, the device would issue a request for each segment of the program instruction set. Needless to say, this would be an exceedingly long process for a spacecraft that is 4 light hours from the earth. SMS will address this issue and provide a suitable solution. ]

The SMS model of the Virtual Spacecraft Device (VSD) describe in section 7 introduces four abstract elements: Executive function, Capabilities, Program Invocations, and Domains. This section describes the services which manage Domains. Domains may be dynamic in nature, coming into existence and being removed from the system either by SMS services or by local action. Services are provided to allow an SMS client to manipulate Domains defined at the SMS server.

- 10.1. THE DOMAIN OBJECT**
- 10.2. INITIATEDOWNSEQUENCE**
- 10.3. DOWNLOADSEGMENT**
- 10.4. TERMINATEDOWNSEQUENCE**
- 10.5. INITIATEUPLOADSEQUENCE**
- 10.6. UPLOADSEGMENT**
- 10.7. TERMINATEUPLOADSEQUENCE**
- 10.8. REQUESTDOMAINDOWNLOAD**
- 10.9. REQUESTDOMAINUPLOAD**
- 10.10. LOADDOMAINCONTENT**
- 10.11. STOREDOMAINCONTENT**
- 10.12. DELETEDOMAIN**
- 10.13. GETDOMAINATTRIBUTES**

## **11. PROGRAM INVOCATION SERVICES**

[Note: Highly sophisticated spacecraft devices are likely to incorporate multiple programs to support a variety of operations. The SMS Program Invocation services are specifically designed to manage the execution and operation of programs on remote devices. Initial inspection of the MMS Program Invocation services suggests that no changes are required to support the space environment. Therefore, it should be possible to employ this portion of the MMS Standard as originally specified. Devices that do not employ multiple programs for operation would not incorporate and support the Program Invocation services. ]

Program Invocation management provides services to control programs running on a VMD.

### **11.1. THE PROGRAM INVOCATION OBJECT**

### **11.2. CREATEPROGRAMINVOCATION**

### **11.3. DELETEPROGRAMINVOCATION**

### **11.4. START**

### **11.5. STOP**

### **11.6. RESUME**

### **11.7. RESET**

### **11.8. KILL**

### **11.9. GETPROGRAMINVOCATIONATTRIBUTES**

## 12.     **VARIABLE ACCESS SERVICES**

[Note: The Variable Access services are extremely powerful services. Nearly all of the MMS implementations provide 90% of their functionality through the application of Variable Access services. Initial inspection of the MMS Variable Access services suggests that only the addition of an unconfirmed WRITE service is needed to adapt the existing MMS Standard for the space environment. ]

Variable Access provides the services to manage variables on VMD.

### **12.1. THE SMS VARIABLE ACCESS OBJECT**

### **12.2. SPECIFICATION OF TYPE**

### **12.3. SPECIFICATION OF ALTERNATE ACCESS**

### **12.4. SPECIFICATION OF DATA VALUES**

### **12.5. SPECIFICATION OF ACCESS TO VARIABLE**

### **12.6. READ**

### **12.7. WRITE**

### **12.8. INFORMATIONREPORT**

### **12.9. GETVARIABLEACCESSATTRIBUTES**

### **12.10. DEFINENAMEDVARIABLE**

### **12.11. DEFINESCATTEREDACCESS**

### **12.12. GETSCATTEREDACCESSATTRIBUTES**

### **12.13. DELETEVARIABLEACCESS**

### **12.14. DEFINENAMEDVARIABLELIST**

### **12.15. GETNAMEDVARIABLELISTATTRIBUTES**

### **12.16. DELETENAMEDVARIABLELIST**

### **12.17. DEFINENAMEDTYPE**

### **12.18. GETNAMEDTYPEATTRIBUTES**

### **12.19. DELETENAMEDTYPE**

### **12.20. CONFORMANCE**

### **12.21. GUIDANCE TO IMPLEMENTORS**

### 13.     **SEMAPHORE MANAGEMENT SERVICES**

[Note: The synchronization, control and coordination of devices (i.e. sub-systems and instruments) operating onboard a spacecraft is critical for the operation of a spacecraft. Successful space missions which achieve optimum results and performance from all of the devices require the careful coordination of all operating elements. In the past, this has been an expensive, time consuming process. The semaphore service provide mechanisms to support the process and reduce to cost of the coordination effort. It is anticipated that semaphore services will be employed to coordinate devices onboard spacecraft or between spacecraft operating in close proximity. Consequently, long delays in communication are not expected and the origin MMS services should address the space environment.]

Semaphore management provides services which allow synchronization, control and coordination of shared resources between VSDs.

#### **13.1. THE SEMAPHORE MANAGEMENT OBJECT**

#### **13.2. TAKECONTROL**

#### **13.3. RELINQUISHCONTROL**

#### **13.4. DEFINESEMAPHORE**

#### **13.5. DELETESEMAPHORE**

#### **13.6. REPORTSEMAPHORESTATUS**

#### **13.7. REPORTPOOLSEMAPHORESTATUS**

#### **13.8. REPORTSEMAPHOREENTRYSTATUS**

#### **13.9. ATTACHTOSEMAPHOREMODIFIER**

#### **13.10. CONFORMANCE**

## **14. OPERATOR COMMUNICATION SERVICES**

[Note: MMS defined services for factory automation making provisions for supervisory personnel to intervene in the process and to communicate via the factory communications systems. While unmanned spacecraft will most likely not use these services, SMS retains these services for their potential application in manned spacecraft. ]

Operator Communications provide services for the exchange of text messages between terminals on board a spacecraft.

### **14.1. THE OPERATOR COMMUNICATIONS MODEL**

#### **14.2. INPUT**

#### **14.3. OUTPUT**

## 15.     **EVENT MANAGEMENT SERVICES**

[Note: The Event Management Services provide a mechanism devices for accessing and managing conditions identified by the device, as an event. As spacecraft become more sophisticated and operate more autonomously, the success of a space mission will become increasingly dependent on robust capabilities to identify, respond and manage anomalous conditions that occur in the course of the mission. The event management model defined in MMS is derived from extensive industrial process control experience and provides a solid foundation for event management of robotic spacecraft. It is anticipated that event services will be employed to coordinate devices onboard spacecraft or between spacecraft operating in close proximity. Consequently, long delays in communication are not expected and the origin MMS services should address the space environment.]

Event management services provide facilities which allow a client MMS-user to define and manage event objects at a VMD and obtain notification of event occurrences.

- 15.1. THE EVENT MANAGEMENT MODEL**
- 15.2. DEFINEEVENTCONDITION**
- 15.3. DELETEEVENTCONDITION**
- 15.4. GETEVENTCONDITIONATTRIBUTES**
- 15.5. REPORTEVENTCONDITIONSTATUS**
- 15.6. ALTEREVENTCONDITIONMONITORING**
- 15.7. TRIGGEREVENT**
- 15.8. DEFINEEVENTACTION**
- 15.9. DELETEEVENTACTION**
- 15.10. GETEVENTACTIONATTRIBUTES**
- 15.11. REPORTEVENTACTIONSTATUS**
- 15.12. DEFINEEVENTENROLLMENT**
- 15.13. DELETEEVENTENROLLMENT**
- 15.14. GETEVENTENROLLMENTATTRIBUTES**
- 15.15. REPORTEVENTENROLLMENTSTATUS**
- 15.16. ALTEREVENTENROLLMENT**
- 15.17. EVENTNOTIFICATION**
- 15.18. ACKNOWLEDGEEVENTENROLLMENT**
- 15.19. GETALARMSUMMARY**
- 15.20. GETALARMENROLLMENTSUMMARY**
- 15.21. ATTACHTOEVENTCONDITIONMODIFIER**
- 15.22. EVENT MANAGEMENT STATE DIAGRAMS**

## **16. JOURNAL MANAGEMENT SERVICES**

[Note: The Journal services are used to create and manage a time based record or log of events and/or data. Each entry in a journal can contain the state of an event, a value of a variable, or character string data. It is anticipated that journal services will be employed in devices onboard the spacecraft. Consequently, long delays in communication are not expected and the origin MMS services should address the space environment.]

Journal management provides services for recording and retrieval of chronologically ordered information concerning events, variable contents of interest in conjunction with events, and textual strings entered by an operator as information.

### **16.1. THE JOURNAL MANAGEMENT MODEL**

### **16.2. READJOURNAL**

### **16.3. WRITEJOURNAL**

### **16.4. INITIALIZEJOURNAL**

### **16.5. REPORTJOURNALSTATUS**

### **16.6. CREATEJOURNAL**

### **16.7. DELETEJOURNAL**

### **16.8. CONFORMANCE REQUIREMENTS UNIQUE TO JOURNAL**

## 17.     **FILE MANAGEMENT SERVICES**

[Note: The File Management services specified by MMS were intended to meet the limited requirements of the manufacturing systems on the factory floor. While the MMS file services provide most of the functions required for the space environment, additional services are required to support a full network file server capability. Examples of services that must be added are: CreateFile, FileWrite, and FileRecordUpdate. These services will be added to SMS. ]

### **17.1. THE SMS FILE MODEL**

### **17.2. FILEOPEN**

### **17.3. FILEREADRECORD**

### **17.4. FILEWRITERECORD**

### **17.5. FILERECORDUPDATE**

### **17.6. FILECLOSE**

### **17.7. FILECOPY**

### **17.8. FILECOPYRESUME**

### **17.9. FILERENAME**

### **17.10. FILEDELETE**

### **17.11. FILEDIRECTORY**

### **17.12. FILEATTRIBUTES**

### **17.13. ADDITIONAL SPECIFICATION FOR CONCLUDE AND ABORT SERVICES**

18.    **ERRORS**

## 19.     **SPECIAL ATTACHMENT 1**

### Tagless Encoding Rules

*The following special attachment was provided by the Electric Power Research Institute (EPRI) based on work underway at the Swiss Federal Institute for Technology for INFORMATION ONLY.*

The notation used for this abstract syntax: TASN

ASN.1 is the notation used by ISO, IEC and CCITT to define application layer protocols, in particular the format of application messages. At the heart of ASN.1 is a fundamental assumption that has important implications on the notation. This assumption is that all ASN.1 types are tagged, i.e. they are associated with a piece of information that allows the identification of the type in its context. In the cases where multiple tags may be conflicting, ASN.1 defines a mechanism to give a type a new tag to make it unique in its context (a sequence, set or a choice).

TASN is described using a notation that differs from ASN.1 essentially by the absence of tags. The acronym TASN stands for Tag-less Abstract Syntax Notation. To solve the problem of optional elements and of choices, the specifier is expected to introduce special fields that indicate the presence or absence of optional elements or that indicate which branch of a choice is taken. As will be seen in the examples given later, the PDUs resulting from this new abstract syntax are much shorter than their equivalent in the traditional abstract syntax. TASN makes use of the same primitive types as ASN.1 with the same notation for these types: BOOLEAN, INTEGER, BIT STRING, OCTET STRING, ...}. Their semantics is exactly the same as in ASN.1.

The main differences concern the SEQUENCE and CHOICE types. As in ASN.1, a SEQUENCE denotes a series of one or more elements which can be optional and that are identified by a name preceding their type. In TASN, when a SEQUENCE contains optional elements, we introduce an “option” element of type BIT STRING whose bits indicate whether the corresponding element is present or not. The following definition in ASN.1:

```
X ::= SEQUENCE {  
    a TypeA OPTIONAL,  
    b TypeB OPTIONAL}
```

is translated in the new notation into:

```
X ::= SEQUENCE {  
    options OptionX,  
    a TypeA OPTIONAL,  
    b TypeB OPTIONAL}
```

```
OptionX = BitString8 {  
    aPresent (0),  
    bPresent (1) }  
(* flags indicating presence of a and b *)
```

## SuperMOCA Space Messaging Service

The syntax itself is longer because additional types must be introduced. However, the corresponding encodings are generally much shorter because they allow to drop the identifier field in tags.

TASN offers a mechanism that is present in ASN.1 and invoked with the DEFINED BY keywords. In TASN the mechanism uses the keywords IDENTIFIED BY. It allows the specifier to associate a type to another one depending on the value of an element defined before. This mechanism is used where CHOICE constructs would be used in ASN.1 specifications. The following example illustrates this mechanism. Suppose we have a sequence composed of two elements, and that the type of the second element may be chosen:

```
X ::= SEQUENCE {  
    element1 Type1,  
    element2 Type2}
```

```
Type2 ::= CHOICE {  
    BOOLEAN,  
    INTEGER}
```

In TASN, due to the absence of tags to identify data elements in choices, we are forced to introduce an additional element in the sequence that will indicate which type is used for element2. Let us call this element "selector". The syntax in TASN is:

```
X = SEQUENCE {  
    selector SelectorType,  
    element1 Type1,  
    element2 Type2};
```

```
SelectorType = Unsigned8 {  
    boolean (0),  
    integer (1)  
}
```

```
Type2 IDENTIFIED BY selector {  
    CASE selector OF  
        0: Type2 = BOOLEAN; break;  
        1: Type2 = INTEGER  
    END };
```

Break is a keyword that is used to separate alternatives of a CASE statement.

In a similar manner, changes are necessary to support the SET type. As MMS does not use this type, we shall therefore terminate our discussion of TASN here.

### **Encoding Rules**

We briefly describe the encoding rules of TASN. These are very similar to encoding rules such as XDR (eXternal Data Representation) -- used by NFS for example -- or the rules defined for the encoding of FMS PDUs. The general idea is that data values do not follow the T, L, V (Type, Length, Value) approach but the V-only approach and sometimes, for example for strings and sequences the L, V approach. The rules for the types used in TASN are the following:

Unsigned8:

Encoded on one octet, most significant bit first.

Unsigned32:

Encoded on four octets whatever the value of the unsigned integer, most significant byte first.

BitString8:

Encoded on one octet, bit 0 first.

VisibleString, all strings:

Encoded with length first and value second. Length field on one or more octets. When most significant bit is one, indicates that following octet(s) encodes the sequel of the length field. Last octet of the length field has most significant bit set to 0.

SEQUENCE

No tag, no length field contrary to ASN.1 BER.

SEQUENCE OF:

Indication of the number of elements in the sequence as an unsigned integer encoded on one or more octets. The number of octets of this number is not known a priori but can be deduced by the fact that each octet indicates whether it is the last (leading bit of 0) or not (leading bit of 1).

### **Encoding Examples**

In all the examples given below, we assume that the invocation identifier is a small number, smaller than 100 and that there are no modifiers.

Status

## SuperMOCA Space Messaging Service

With the classical MMS abstract syntax, using ASN.1 Basic Encoding Rules, the Status request is encoded in 8 octets:

```
A0 06      confirmedRequest
02 01 yy    InvokeID
80 01 00    Status, no extendedDerivation
```

With the new abstract syntax and the new encoding rules, the request is encoded in 3 octets:

```
00 Status
00 flags(ModifiersPresent = FALSE,
        CS-Detail = FALSE,
        extendedDerivation = FALSE)
yy InvokeID
```

Read

With the classical MMS abstract syntax, using ASN.1 Basic Encoding Rules, the request is encoded in 23 octets:

```
A0 15      confirmedRequest
02 01 yy    InvokeID
A4 10      Read
A1 0E      varAccessSpecification
A0 0C      listOfVariable
30 0A      SEQUENCE
A0 08      name
80 06      vmd-specific name
AA BB      actual name
CC DD
EE FF
```

With TASN and the new encoding rules, the request is encoded in 13 octets:

```
04 Read
xx flags(no modifiers,
        variableList, no Spec
        With Result}
yy invokeID
01 SEQUENCE OF 1 element
00 access = name
00 vmd-specific name
06 name length = 6
```

## SuperMOCA Space Messaging Service

AA BB actual name  
CC DD  
EE FF

Comparison of length between classical ASN.1 BER and TASN

Service	BER (A)	TASN (B)	Ratio B/A (%)
Status Req	8	3	37.5
Status Rsp	13	4	30.8
Identify Req	7	3	42.8
Identify Rsp	32	24	75
Start Req	16	10	62.5
Start Rsp	8	3	37.5
Stop Req	16	10	62.5
Stop Rsp	16	3	18.75
Read 1 Req	23	13	56.5
Read 1 Rsp	12	7	58.3
Read Array 100 Req	23	13	56.5
Read Array 100 Rsp	415	304	73.25
Read List 100 Req	1219	404	33.1
Read List 100 Rsp	419	304	73.25
Write 1 Req	27	17	63
Write 1 Rsp	9	5	55.5
Write List 100 Req	1619	1205	74.4
Write List 100 Rsp	209	104	49.7

Comparison between classical BER and TASN: PDU lengths in octets.

Req = Request PDU, Rsp = Response PDU

*The preceding special attachment was provided by the Electric Power Research Institute (EPRI) based on work underway at the Swiss Federal Institute for Technology for INFORMATION ONLY.*